TABLE OF CONTENTS

Chapter 1. NVML API Reference ................................................................. 1
Chapter 2. Known Issues ....................................................................... 3
Chapter 3. Change Log .......................................................................... 4
Chapter 4. Modules ............................................................................... 9
  4.1. Device Structs .............................................................................. 9
      nvmlPciInfo_t ............................................................................... 10
      nvmlEccErrorCounts_t ............................................................... 10
      nvmlUtilization_t ....................................................................... 10
      nvmlMemory_t ........................................................................... 10
      nvmlBAR1Memory_t ................................................................... 10
      nvmlProcessInfo_t ..................................................................... 10
      nvmlNVLinkUtilizationControl_t .............................................. 10
      nvmlBridgeChipInfo_t ................................................................ 10
      nvmlBridgeChipHierarchy_t ..................................................... 10
      nvmlValue_t ............................................................................. 10
      nvmlSample_t ........................................................................... 10
      nvmlViolationTime_t .................................................................. 10
      nvmlBridgeChipType_t ................................................................ 10
      nvmlNVLinkUtilizationCountUnits_t .......................................... 10
      nvmlNVLinkUtilizationCountPktTypes_t ..................................... 11
      nvmlNVLinkCapability_t ............................................................ 11
      nvmlNVLinkErrorCounter_t ....................................................... 11
      nvmlGpuTopologyLevel_t ........................................................... 12
      nvmlSamplingType_t ................................................................... 12
      nvmlPcieUtilCounter_t ............................................................... 12
      nvmlValueType_t ......................................................................... 13
      nvmlPerfPolicyType_t ............................................................... 13
      NVML_VALUE_NOT_AVAILABLE ............................................... 13
      NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE .................................. 13
      NVML_NVLINK_MAX_LANES ...................................................... 13
      NVML_MAX_PHYSICAL_BRIDGE ................................................. 13
  4.2. Device Enums ............................................................................... 13
      nvmlEnableState_t ..................................................................... 14
      nvmlBrandType_t ....................................................................... 14
      nvmlTemperatureThresholds_t ............................................... 14
      nvmlTemperatureSensors_t ....................................................... 14
      nvmlComputeMode_t ................................................................... 15
      nvmlMemoryErrorType_t ........................................................... 15
      nvmlEccCounterType_t .............................................................. 15
      nvmlClockType_t ........................................................................ 16
NVML\_DEVICE\_NAME\_BUFFER\_SIZE..............................................................................30
NVML\_DEVICE\_SERIAL\_BUFFER\_SIZE..................................................................... 30
NVML\_DEVICE\_VBIOS\_VERSION\_BUFFER\_SIZE............................................................30

4.9. System Queries
nvmlSystemGetDriverVersion............................................................................. 30
nvmlSystemGetNVMLVersion.............................................................................. 31
nvmlSystemGetProcessName.............................................................................. 32

4.10. Unit Queries
nvmlUnitGetCount..........................................................................................33
nvmlUnitGetHandleByIndex............................................................................. 33
nvmlUnitGetUnitInfo....................................................................................... 34
nvmlUnitGetLedState...................................................................................... 34
nvmlUnitGetPsuInfo....................................................................................... 35
nvmlUnitGetTemperature................................................................................. 36
nvmlUnitGetFanSpeedInfo.................................................................................36
nvmlUnitGetDevices........................................................................................37
nvmlSystemGetHicVersion.................................................................................38

4.11. Device Queries
nvmlDeviceGetCount........................................................................................39
nvmlDeviceGetHandleByIndex............................................................................ 39
nvmlDeviceGetHandleBySerial............................................................................41
nvmlDeviceGetHandleByUUID...............................................................................42
nvmlDeviceGetHandleByPciBusId.......................................................................43
nvmlDeviceGetName....................................................................................... 44
nvmlDeviceGetBrand....................................................................................... 44
nvmlDeviceGetIndex....................................................................................... 45
nvmlDeviceGetSerial....................................................................................... 46
nvmlDeviceGetCpuAffinity............................................................................... 47
nvmlDeviceSetCpuAffinity............................................................................... 47
nvmlDeviceClearCpuAffinity.............................................................................. 48
nvmlDeviceGetTopologyCommonAncestor..............................................................49
nvmlDeviceGetTopologyNearestGpus...................................................................49
nvmlSystemGetTopologyGpuSet..........................................................................50
nvmlDeviceGetUUID........................................................................................ 51
nvmlDeviceGetMinorNumber..............................................................................52
nvmlDeviceGetBoardPartNumber.........................................................................52
nvmlDeviceGetInforomVersion............................................................................53
nvmlDeviceGetInforomImageVersion....................................................................54
nvmlDeviceGetInforomConfigurationChecksum........................................................55
nvmlDeviceValidateInforom...............................................................................56
nvmlDeviceGetDisplayMode............................................................................... 56
nvmlDeviceGetDisplayActive..............................................................................57
nvmlDeviceGetPersistenceMode..........................................................................58
nvmlDeviceGetVbiosVersion............................................................................... 93
nvmlDeviceGetBridgeChipInfo............................................................................ 93
nvmlDeviceGetComputeRunningProcesses.............................................................. 94
nvmlDeviceGetGraphicsRunningProcesses.............................................................. 95
nvmlDeviceOnSameBoard..................................................................................96
nvmlDeviceGetAPIRestriction............................................................................. 97
nvmlDeviceGetSamples.................................................................................... 98
nvmlDeviceGetBAR1MemoryInfo..........................................................................99
nvmlDeviceGetViolationStatus.......................................................................... 100
nvmlDeviceGetRetiredPages............................................................................. 101
nvmlDeviceGetRetiredPagesPendingStatus............................................................ 102
4.12. Unit Commands....................................................................................... 102
nvmlUnitSetLedState..................................................................................... 102
4.13. Device Commands.................................................................................... 103
nvmlDeviceSetPersistenceMode......................................................................... 103
nvmlDeviceSetComputeMode............................................................................104
nvmlDeviceSetEccMode...................................................................................105
nvmlDeviceClearEccErrorCounts........................................................................ 106
nvmlDeviceSetDriverModel...............................................................................107
nvmlDeviceSetApplicationsClocks...................................................................... 108
nvmlDeviceSetPowerManagementLimit................................................................ 109
nvmlDeviceSetGpuOperationMode......................................................................110
nvmlDeviceSetAPIRestriction............................................................................ 111
4.14. NvLink Methods....................................................................................... 112
nvmlDeviceGetNvLinkState.............................................................................. 112
nvmlDeviceGetNvLinkVersion............................................................................113
nvmlDeviceGetNvLinkCapability........................................................................ 113
nvmlDeviceGetNvLinkRemotePciInfo...................................................................114
nvmlDeviceGetNvLinkErrorCounter.....................................................................115
nvmlDeviceSetNvLinkUtilizationControl............................................................... 116
nvmlDeviceGetNvLinkUtilizationCounter.............................................................. 116
nvmlDeviceFreezeNvLinkUtilizationCounter.......................................................... 117
nvmlDeviceResetNvLinkUtilizationCounter............................................................118
4.15. Event Handling Methods..........................................................................118
nvmlEventData_t.......................................................................................... 118
Event Types................................................................................................ 118
nvmlEventSet_t............................................................................................ 118
nvmlEventSetCreate...................................................................................... 119
nvmlDeviceRegisterEvents............................................................................... 119
nvmlDeviceGetSupportedEventTypes...................................................................120
nvmlEventSetWait......................................................................................... 121
nvmlEventSetFree......................................................................................... 122
4.15.1. Event Types...................................................................................... 122
4.16. Grid Queries........................................................................................................124

4.17. GRID Commands..................................................................................................125

4.18. NvmlClocksThrottleReasons................................................................................125

Chapter 5. Data Structures..........................................................................................128

nvmlAccountingStats_t.............................................................................................128

gpuUtilization...........................................................................................................129

memoryUtilization....................................................................................................129

maxMemoryUsage.....................................................................................................129

time........................................................................................................................129

startTime...............................................................................................................130

isRunning...............................................................................................................130

reserved...............................................................................................................130

nvmlBAR1Memory_t....................................................................................................130

bar1Total.................................................................................................................130

bar1Free..................................................................................................................130

bar1Used..................................................................................................................130

nvmlBridgeChipHierarchy_t.....................................................................................130

bridgeCount..........................................................................................................131

bridgeChipInfo......................................................................................................131

nvmlBridgeChipInfo_t..............................................................................................131

type.......................................................................................................................131

fwVersion..............................................................................................................131

nvmlEccErrorCounts_t.............................................................................................131

l1Cache...................................................................................................................132

l2Cache...................................................................................................................132

deviceMemory........................................................................................................132
Chapter 6. Data Fields....................................................................................... 138
Chapter 7. Deprecated List............................................................................... 142
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:
- Linux: 32-bit and 64-bit

Supported products:
- Full Support
  - NVIDIA Tesla Line:
    - S2050, C2050, C2070, C2075,
    - M2050, M2070, M2075, M2090,
    - X2070, X2090,
    - K8, K10, K20, K20X, K20Xm, K20c, K20m, K20s, K40c, K40m, K40t, K40s, K40st, K40d, K80
  - NVIDIA Quadro Line:
    - 410, 600, 2000, 4000, 5000, 6000, 7000, M2070-Q
    - K2000, K2000D, K4000, K5000, K6000
  - NVIDIA GRID Line:
    - K1, K2, K340, K520
  - NVIDIA GeForce Line: None
Limited Support

- NVIDIA Tesla Line: S1070, C1060, M1060 and all other 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 `%ProgramW6432%"NVIDIA Corporation"\NVSMI` on Windows, but will not be added to the path. 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 will be found on the standard library path. For 64-bit Linux, both the 32-bit and 64-bit NVML libraries will be installed.

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

List of changes can be found in the Change Log.
Chapter 2.
KNOWN ISSUES

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

- On Linux when X Server is running `nvmlDeviceGetComputeRunningProcesses` may return a `nvmlProcessInfo_t::usedGpuMemory` value that is larger than the actual value. This will be fixed in a future release.
- 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. Next 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 few MHz.
Chapter 3.
CHANGE LOG

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

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 `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.

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:

- Device Structs
- Device Enums
- GRID Enums
- Unit Structs
- Accounting Statistics
- Initialization and Cleanup
- Error reporting
- Constants
- System Queries
- Unit Queries
- Device Queries
- Unit Commands
- Device Commands
- NvLink Methods
- Event Handling Methods
  - Event Types
- Grid Queries
- GRID Commands
- NvmlClocksThrottleReasons

4.1. Device Structs
struct nvmlPciInfo_t
struct nvmlEccErrorCounts_t
struct nvmlUtilization_t
struct nvmlMemory_t
struct nvmlBAR1Memory_t
struct nvmlProcessInfo_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_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

enum nvmlNvLinkCapability_t
Enum to represent NvLink queryable capabilities

Values

NVML_NVLINK_CAP_P2P_SUPPORTED = 0
NVML_NVLINK_CAP_SYSTEM_ACCESS = 1
NVML_NVLINK_CAP_P2P_ATOMICS = 2
NVML_NVLINK_CAP_SYSTEM_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_COUNT
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_CPU = 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_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_COUNT

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

Values
NVML_PERF_POLICY_POWER = 0
NVML_PERF_POLICY_THERMAL = 1
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 16
Buffer size guaranteed to be large enough for pci bus id

#define NVML_NVLINK_MAX_LANES 4
Maximum number of NvLink lanes supported

#define NVML_MAX_PHYSICAL_BRIDGE (128)
Maximum limit on Physical Bridges per Board

4.2. Device Enums
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_COUNT

enum nvmlTemperatureThresholds_t

Temperature thresholds.

Values

NVML_TEMPERATURE_THRESHOLD_SHUTDOWN = 0
NVML_TEMPERATURE_THRESHOLD_SLOWDOWN = 1
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_COUNT

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

\texttt{NVML\_GOM\_LOW\_DP = 2}

Designed for running graphics applications that don't require high bandwidth double precision

\textbf{enum nvmlInforomObject\_t}

Available infoROM objects.

\textbf{Values}

\texttt{NVML\_INFOROM\_OEM = 0}

An object defined by OEM.

\texttt{NVML\_INFOROM\_ECC = 1}

The ECC object determining the level of ECC support.

\texttt{NVML\_INFOROM\_POWER = 2}

The power management object.

\texttt{NVML\_INFOROM\_COUNT}

This counts the number of infoROM objects the driver knows about.

\textbf{enum nvmlReturn\_t}

Return values for NVML API calls.

\textbf{Values}

\texttt{NVML\_SUCCESS = 0}

The operation was successful.

\texttt{NVML\_ERROR\_UNINITIALIZED = 1}

NVML was not first initialized with nvmlInit().

\texttt{NVML\_ERROR\_INVALID\_ARGUMENT = 2}

A supplied argument is invalid.

\texttt{NVML\_ERROR\_NOT\_SUPPORTED = 3}

The requested operation is not available on target device.

\texttt{NVML\_ERROR\_NO\_PERMISSION = 4}

The current user does not have permission for operation.

\texttt{NVML\_ERROR\_ALREADY\_INITIALIZED = 5}

Deprecated: Multiple initializations are now allowed through ref counting.

\texttt{NVML\_ERROR\_NOT\_FOUND = 6}

A query to find an object was unsuccessful.

\texttt{NVML\_ERROR\_INSUFFICIENT\_SIZE = 7}

An input argument is not large enough.

\texttt{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_UNKNOWN = 999
   An internal driver error occurred.

enum nvmlMemoryLocation_t

Memory locations

See nvmlDeviceGetMemoryErrorCounter

Values

NVML_MEMORY_LOCATION_L1_CACHE = 0
   GPU L1 Cache.

NVML_MEMORY_LOCATION_L2_CACHE = 1
   GPU L2 Cache.

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_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.

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 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. GRID Enums

define 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-Passthrough.
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.

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.

```c
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`

```c
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 the device doesn’t support this feature or accounting mode is disabled
- 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 the device 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 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.

See nvmlDeviceGetAccountingMode  See nvmlDeviceGetAccountingStats  See nvmlDeviceClearAccountingPids

```c
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. Initialization and Cleanup

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

**nvmlReturn_t nvmlInit (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.

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.

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 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()`.
For all products.
This method should be called after NVML work is done, once for each call to `nvmlInit()`. 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()`.

4.7. Error reporting
This chapter describes helper functions for error reporting routines.

```c
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.

4.8. Constants

```c
#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 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 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

4.9. 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 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.

4.10. 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.

```c
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.

```c
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.

\texttt{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.
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.

4.11. 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(), nvmlDeviceGetHandleBySerial(), nvmlDeviceGetHandleByPciBusId(). or nvmlDeviceGetHandleByUUID().
nvmlReturn_t nvmlDeviceGetCount (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 nvmlDeviceGetHandleByIndex (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()`. 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()`.

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

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 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 (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().

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 64 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE.

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.

```c
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()`. 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()` and `nvmlDeviceGetHandleByUUID()`.

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 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

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 a device using the guidelines given in `nvmlDeviceGetCpuAffinity()` Currently supports up to 64 processors.

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

```c
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 Kepler or newer fully supported devices. Supported on Linux only.
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 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.

\texttt{nvmlReturn_t nvmlDeviceGetUUID (nvmlDevice_t device, char *uuid, unsigned int length)}

Parameters
\textbf{device}
The identifier of the target device
\textbf{uuid}
Reference in which to return the GPU UUID
\textbf{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 80 characters in length (including the NULL terminator). See \texttt{nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE}.
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 device board part number which is programmed into the board's InfoROM

For all products.

```c
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 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 nvmlDeviceGetPciInfo (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 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 on virtualized GPU environments.

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 and rollover have been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or value or rollover are 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 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 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`.

See also:

`nvmlDeviceGetApplicationsClock`

`nvmlDeviceSetApplicationsClocks`

For Fermi or newer non-GeForce fully supported devices and Maxwell or newer GeForce devices.

```c
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`

```c
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`

```c
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.

`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.

`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.
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 percent of the maximum, i.e. full speed is 100%.

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.

```c
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.

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

Parameters

device
   The identifier of the target device
clocksThrottleReasons
   Reference in which to return bitmask of active clocks throttle reasons

Returns

- NVML_SUCCESS if clocksThrottleReasons has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clocksThrottleReasons 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 throttling 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:

NvmlClocksThrottleReasons
nvmlDeviceGetSupportedClocksThrottleReasons

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

Parameters

device
  The identifier of the target device

supportedClocksThrottleReasons
  Reference in which to return bitmask of supported clocks throttle reasons

Returns

- NVML_SUCCESS if supportedClocksThrottleReasons has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or supportedClocksThrottleReasons 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 throttle reasons that can be returned by nvmlDeviceGetCurrentClocksThrottleReasons

For all fully supported products.

This method is not supported on virtualized GPU environments.

See also:

NvmlClocksThrottleReasons

nvmlDeviceGetCurrentClocksThrottleReasons

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.

```c
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 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.

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

**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.

```c
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_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 and total memory available on the device, in bytes.

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_t for details on available memory info.

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()`

```c
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 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
derrorType
   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
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, bitType, 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.

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.

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.

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.

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(nvmlDevice_t device, unsigned int *infoCount, nvmlProcessInfo_t *infos)
```

Parameters

- `device`: The identifier of the target device
- `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_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.

See also:
nvmlSystemGetProcessName

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

Parameters
device
   The identifier of the target device
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_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.

See also:
nvmlSystemGetProcessName

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.

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).

For Kepler or newer fully supported devices.

```c
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 trying 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 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 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.

4.12. 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()

4.13. 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.

\[ \text{nvmlReturn_t } \text{nvmIDeviceSetPersistenceMode} (\text{nvmIDevice_t device, nvmIEnableState_t mode}) \]

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.

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

See [nvmlComputeMode_t](https://docs.nvidia.com/nvidia-driver/api-guide/html/group__NVML__types.html) for details on available compute modes.

**See also:**

nvmlDeviceGetComputeMode()

```c
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
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()

\textbf{nvmlReturn\_t nvmlDeviceSetApplicationsClocks}\n\textbf{(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

\textbf{\textbullet{} NVML\_SUCCESS} if new settings were successfully set
\textbf{\textbullet{} 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.

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 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`

```c
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 restrictions 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

4.14. NvLink Methods
This chapter describes methods that NVML can perform on NVLINK enabled devices.

nvmlReturn_t nvmlDeviceGetNvLinkState (nvmlDevice_t device, unsigned int lane, nvmlEnableState_t *isActive)

Parameters
- device
  The identifier of the target device
- lane
  Specifies the NvLink lane to be queried
- isActive
  nvmlEnableState_t where NVML_FEATURE_ENABLED indicates that the lane 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 lane 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 lane specified
For newer than Maxwell fully supported devices.

`nvmlReturn_t nvmlDeviceGetNvLinkVersion`
` (nvmlDevice_t device, unsigned int lane, unsigned int *version)`

Parameters

- **device**
  The identifier of the target device
- **lane**
  Specifies the NvLink lane 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 lane 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 lane specified
For newer than Maxwell fully supported devices.

`nvmlReturn_t nvmlDeviceGetNvLinkCapability`
` (nvmlDevice_t device, unsigned int lane, `
` nvmlNvLinkCapability_t capability, unsigned int *capResult)`

Parameters

- **device**
  The identifier of the target device
lane
   Specifies the NvLink lane 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, lane, 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 lane 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 newer than Maxwell fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkRemotePciInfo(nvmlDevice_t device, unsigned int lane, nvmlPciInfo_t *pci)

Parameters

device
   The identifier of the target device

lane
   Specifies the NvLink lane to be queried

pci
   nvmlPciInfo_t of the remote node for the specified lane

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 lane 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 lane. Note: pciSubSystemId is not filled in this function and is indeterminate.
For newer than Maxwell fully supported devices.

`nvmlReturn_t nvmlDeviceGetNvLinkErrorCounter(nvmlDevice_t device, unsigned int lane, nvmlNvLinkErrorCounter_t counter, unsigned long long *counterValue)`

Parameters
- **device**
  - The identifier of the target device
- **lane**
  - Specifies the NvLink lane 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, lane, 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 newer than Maxwell fully supported devices.
nvmlReturn_t nvmlDeviceSetNvLinkUtilizationControl
(nvmlDevice_t device, unsigned int lane,
nvmlNvLinkUtilizationControl_t *control)

Parameters

device
The identifier of the target device

lane
Specifies the NvLink lane to be queried

control
A reference to the nvmlNvLinkUtilizationControl_t to set

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 or lane or control is invalid
‣ NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description
Set the NVLINK utilization counter control information Please refer to
nvmlNvLinkUtilizationControl_t for the structure definition
For newer than Maxwell fully supported devices.

nvmlReturn_t nvmlDeviceGetNvLinkUtilizationCounter
(nvmlDevice_t device, unsigned int lane, unsigned long
long *rxcounter, unsigned long long *txcounter)

Parameters

device
The identifier of the target device

lane
Specifies the NvLink lane to be queried

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 or lane 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

Retrieve the NVLINK utilization counter based on the current control. In general, it is good practice to use `nvmlDeviceSetNvLinkUtilizationControl` before reading the utilization counters as they have no default state.

For newer than Maxwell fully supported devices.

```c
nvmlReturn_t
nvmlDeviceFreezeNvLinkUtilizationCounter
(nvmlDevice_t device, unsigned int lane,
 nvmlEnableState_t freeze)
```

Parameters

- **device**: The identifier of the target device
- **lane**: Specifies the NvLink lane to be queried
- **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, lane, or freeze is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Freeze the NVLINK utilization counters. Both the receive and transmit counters are operated on by this function.
For newer than Maxwell fully supported devices.

```
nvmlReturn_t nvmlDeviceResetNvLinkUtilizationCounter(nvmlDevice_t device, unsigned int lane)
```

**Parameters**

- **device**
  The identifier of the target device

- **lane**
  Specifies the NvLink lane to be queried

**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, lane is invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

Reset the NVLINK utilization counters. Both the receive and transmit counters are operated on by this function.

For newer than Maxwell fully supported devices.

### 4.15. 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`

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`

```c
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 (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
- timeouts
  - 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)

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.

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

4.15.1. 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`

```c
#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

```c
#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

```c
#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.

```c
#define nvmlEventTypeXidCriticalError 0x0000000000000008LL
Event that Xid critical error occurred.
```

```c
#define nvmlEventTypeClock 0x0000000000000010LL
Event about clock changes.
```

Kepler only

```c
#define nvmlEventTypeNone 0x0000000000000000LL
Mask with no events.
```
#define nvmlEventTypeAll (nvmlEventTypeNone \ | nvmlEventTypeSingleBitEccError \ | nvmlEventTypeDoubleBitEccError \ | nvmlEventTypePState \ | nvmlEventTypeClock \ | nvmlEventTypeXidCriticalError \ )

Mask of all events.

## 4.16. Grid Queries

This chapter describes NVML operations that are associated with NVIDIA GRID products.

```c
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.

KEPLER_OR_NEWER%
4.17. GRID Commands

This chapter describes NVML operations that are associated with NVIDIA GRID products.

```
nvmlReturn_t nvmlDeviceSetVirtualizationMode
  (nvmlDevice_t device, nvmlGpuVirtualizationMode_t_t virtualMode)
```

**Parameters**

- **device**
  - Identifier of the target device
- **virtualMode**
  - Virtualization mode. One of NVML_GPU_VIRTUALIZATION_?

**Returns**

- NVML_SUCCESS if pVirtualMode is set
- 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_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.

4.18. NvmlClocksThrottleReasons

```
#define nvmlClocksThrottleReasonGpuIdle
  0x0000000000000001LL
```

Nothing is running on the GPU and the clocks are dropping to Idle state
This limiter may be removed in a later release

```c
#define nvmlClocksThrottleReasonApplicationsClocksSetting
0x0000000000000002LL
```

GPU clocks are limited by current setting of applications clocks

See also:
- `nvmlDeviceSetApplicationsClocks`
- `nvmlDeviceGetApplicationsClock`

```c
#define nvmlClocksThrottleReasonUserDefinedClocks	nvmlClocksThrottleReasonApplicationsClocksSetting
```

Deprecated Renamed to `nvmlClocksThrottleReasonApplicationsClocksSetting` as the name describes the situation more accurately.

```c
#define nvmlClocksThrottleReasonSwPowerCap
0x0000000000000004LL
```

SW Power Scaling algorithm is reducing the clocks below requested clocks

See also:
- `nvmlDeviceGetPowerUsage`
- `nvmlDeviceSetPowerManagementLimit`
- `nvmlDeviceGetPowerManagementLimit`

```c
#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`

```c
#define nvmlClocksThrottleReasonUnknown 0x8000000000000000LL
```

Some other unspecified factor is reducing the clocks

```c
#define nvmlClocksThrottleReasonNone 0x0000000000000000LL
```

Bit mask representing no clocks throttling

Clocks are as high as possible.

```c
#define nvmlClocksThrottleReasonAll (nvmlClocksThrottleReasonNone | nvmlClocksThrottleReasonGpuIdle | nvmlClocksThrottleReasonApplicationsClocksSetting | nvmlClocksThrottleReasonSwPowerCap | nvmlClocksThrottleReasonHwSlowdown | nvmlClocksThrottleReasonUnknown)
```

Bit mask representing all supported clocks throttling reasons New reasons might be added to this list in the future
Chapter 5.
DATA STRUCTURES

Here are the data structures with brief descriptions:

- `nvmlAccountingStats_t`
- `nvmlBAR1Memory_t`
- `nvmlBridgeChipHierarchy_t`
- `nvmlBridgeChipInfo_t`
- `nvmlEccErrorCounts_t`
- `nvmlEventData_t`
- `nvmlHwbcEntry_t`
- `nvmlLedState_t`
- `nvmlMemory_t`
- `nvmlNvLinkUtilizationControl_t`
- `nvmlPciInfo_t`
- `nvmlProcessInfo_t`
- `nvmlPSUInfo_t`
- `nvmlSample_t`
- `nvmlUnitFanInfo_t`
- `nvmlUnitFanSpeeds_t`
- `nvmlUnitInfo_t`
- `nvmlUtilization_t`
- `nvmlValue_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_NOTAVAILABLE 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_NOTAVAILABLE 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_NOTAVAILABLE 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
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. 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.

5.6. 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 last XID error for the device in the event of nvmlEventTypeXidCriticalError.

5.7. nvmlHwbcEntry_t Struct Reference

Description of HWBC entry

5.8. 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.

5.9. nvmlMemory_t Struct Reference
Memory allocation information for a device.

unsigned long long nvmlMemory_t::total
Total installed FB memory (in bytes).

unsigned long long nvmlMemory_t::free
Unallocated FB memory (in bytes).

unsigned long long nvmlMemory_t::used
Allocated FB memory (in bytes). Note that the driver/GPU always sets aside a small amount of memory for bookkeeping.

5.10. nvmlNvLinkUtilizationControl_t Struct Reference
Struct to define the NVLINK counter controls

5.11. nvmlPciInfo_t Struct Reference
PCI information about a GPU device.
char nvmlPciInfo_t::busId
The 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 0xffff.

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.

5.12. 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.

5.13. 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).

### 5.14. nvmlSample_t Struct Reference

Information for Sample

`unsigned long long nvmlSample_t::timeStamp`

CPU Timestamp in microseconds.

`nvmlSample_t::sampleValue`

Sample Value.

### 5.15. 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.

5.16. 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.

5.17. 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.

5.18. 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.
Data Structures

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.

5.19. nvmlValue_t Union Reference
Union to represent different types of Value

double nvmlValue_t::dVal
If the value is double.

unsigned int nvmlValue_t::uiVal
If the value is unsigned int.

unsigned long nvmlValue_t::ulVal
If the value is unsigned long.

unsigned long long nvmlValue_t::ullVal
If the value is unsigned long long.

5.20. 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:

B
bar1Free
    nvmlBAR1Memory_t
bar1Total
    nvmlBAR1Memory_t
bar1Used
    nvmlBAR1Memory_t
bridgeChipInfo
    nvmlBridgeChipHierarchy_t
bridgeCount
    nvmlBridgeChipHierarchy_t
bus
    nvmlPciInfo_t
busId
    nvmlPciInfo_t

C
cause
    nvmlLedState_t
color
    nvmlLedState_t
count
    nvmlUnitFanSpeeds_t
current
    nvmlPSUInfo_t
D
  device
    nvmlPciInfo_t
    nvmlEventData_t
  deviceMemory
    nvmlEccErrorCounts_t
  domain
    nvmlPciInfo_t
  dVal
    nvmlValue_t

E
  eventData
    nvmlEventData_t
  eventType
    nvmlEventData_t

F
  fans
    nvmlUnitFanSpeeds_t
  firmwareVersion
    nvmlUnitInfo_t
  free
    nvmlMemory_t
  fwVersion
    nvmlBridgeChipInfo_t

G
  gpu
    nvmlUtilization_t
  gpuUtilization
    nvmlAccountingStats_t

I
  id
    nvmlUnitInfo_t
  isRunning
    nvmlAccountingStats_t

L
  l1Cache
    nvmlEccErrorCounts_t
Data Fields

l2Cache
  nvmlEccErrorCounts_t

M
maxMemoryUsage
  nvmlAccountingStats_t
memory
  nvmlUtilization_t
memoryUtilization
  nvmlAccountingStats_t

N
name
  nvmlUnitInfo_t

P
pciDeviceId
  nvmlPciInfo_t
pciSubSystemId
  nvmlPciInfo_t
pid
  nvmlProcessInfo_t
power
  nvmlPSUInfo_t

R
referenceTime
  nvmlViolationTime_t
registerFile
  nvmlEccErrorCounts_t
reserved
  nvmlAccountingStats_t

S
sampleValue
  nvmlSample_t
serial
  nvmlUnitInfo_t
speed
  nvmlUnitFanInfo_t
startTime
  nvmlAccountingStats_t
Data Fields

state
   nvmlPSUInfo_t
   nvmlUnitFanInfo_t

T
   time
      nvmlAccountingStats_t
timeStamp
   nvmlSample_t
total
   nvmlMemory_t
type
   nvmlBridgeChipInfo_t

U
   uiVal
      nvmlValue_t
   ullVal
      nvmlValue_t
   ulVal
      nvmlValue_t
   used
      nvmlMemory_t
   usedGpuMemory
      nvmlProcessInfo_t

V
   violationTime
      nvmlViolationTime_t
   voltage
      nvmlPSUInfo_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 `nvmlDeviceGetDetailedEccErrors`
This API supports only a fixed set of ECC error locations. On different GPU architectures, different locations are supported. See `nvmlDeviceGetMemoryErrorCounter`.
Global nvmlClocksThrottleReasonUserDefinedClocks

Renamed to nvmlClocksThrottleReasonApplicationsClocksSetting as the name describes the situation more accurately.
ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

© 2007-2018 NVIDIA Corporation. All rights reserved.