# NVML API REFERENCE MANUAL

May 16, 2012

Version 3.295.54

# Contents

# Chapter 1

# NVML API Reference

The NVIDIA Management Library (NVML) is a C-based programmatic interface for monitoring and managing various states within NVIDIA Tesla ™GPUs.

It is intended to be a platform for building 3rd party applications, and is also the underlying library for the NVIDIA-supported nvidia-smi tool.

NVML is thread-safe so it is safe to make simultaneous NVML calls from multiple threads.

### API Documentation

Supported OS platforms:

- Windows: Windows Server 2008 R2 64bit, Windows 7 64bit
- Linux: 32-bit and 64-bit

Supported products:

- Full Support
  - NVIDIA Tesla ™Line: S1070, S2050, C1060, C2050/70/75, M2050/70/75/90, X2070/90
  - NVIDIA Quadro ®Line: 4000, 5000, 6000, 7000, M2070-Q
  - NVIDIA GeForce ®Line: None
- Limited Support
  - NVIDIA Tesla ™Line: None
  - 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 Changelog

## 1.1 Feature Matrix

| Queries | | S1070 | S2050 |
|---|---|---|---|
| Product Id | | ✓ | ✓ |
| Serial Number | | ✓ | ✓ |
| Firmware Version | | ✓ | ✓ |
| Attached GPUs | | ✓ | ✓ |
| LED State | Color | ✓ | ✓ |
| | Cause | ✓ | ✓ |
| Temperature | Intake | ✓ | ✓ |
| | Exhaust | ✗ | ✗ |
| | Board | ✗ | ✗ |
| PSU | PSU State | ✓ | ✓ |
| | Voltage | ✓ | ✓ |
| | Current | ✓ | ✓ |
| Fans | Fan Speed | ✓ | ✓ |
| | Fan State | ✓ | ✓ |

| Commands | S1070 | S2050 |
|---|---|---|
| Toggle LED State | ✓ | ✓ |

Figure 1.1: This chart shows which unit-level features are available for each S-class product. All GPUs within each S-class product also provide the information listed in the Device chart below.

| Queries | | | C2050 | C2070 | C2075 | M2050 | M2070 | M2075 | M2090 | S2050 | X2070 | X2090 | Gemini |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Board Serial Number | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| GPU UUID | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| VBios Version | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Inforom Version | OEM Object | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | ECC Object | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Power Object | | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ |
| PCI Info | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Compute Mode | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Display Mode | | | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Persistence Mode (Linux-Only) | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ECC Mode | Current | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Pending | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Driver Model (Win7-Only) | Current | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Pending | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Fan Speed | | | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| GPU Temperature | | | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ |
| Memory Usage | Total | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Used | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Free | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Power Readings | Power Management Enabled | | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✗ |
| | Current Power Draw | | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ |
| | Power Draw Limit | | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ | ✓ |
| Clock Speeds | Current | Graphics | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | SM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | Memory | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Max | Graphics | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | SM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | Memory | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Utilization Rates | GPU Compute | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | PCIE Memory | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ECC Errors | Volatile | Location-Based | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | Total | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Aggregate | Location-Based | ✗ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| | | Total | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Performance State | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Process Info | Process Id | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Process Name | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Allocated Device Memory | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

| Commands | C2050 | C2070 | C2075 | M2050 | M2070 | M2075 | M2090 | S2050 | X2070 | X2090 | Gemini |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Set Compute Mode | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Set Persistence Mode (Linux-Only) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Set Display Model (Win7-Only) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Set ECC Mode | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Clear ECC Errors | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Reset GPU (Linux-Only) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Figure 1.2: This chart shows which features are available for each Fermi and Kepler architecture GPU product.

| Queries | | | 4000 | 5000 | 6000 | Quadro Plex 7000 | M2070-Q | 600 | 2000 | 3000M | 410 | S1070 | C1060 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Board Serial Number | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| GPU UUID | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| VBios Version | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Inforom Version | OEM Object | | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | ECC Object | | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | Power Object | | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| PCI Info | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Compute Mode | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Display Mode | | | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| Persistence Mode (Linux-Only) | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ECC Mode | Current | | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | Pending | | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Driver Model (Win7-Only) | Current | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Pending | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Fan Speed | | | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ |
| GPU Temperature | | | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Memory Usage | Total | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Used | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Free | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Power Readings | Power Management Enabled | | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | Current Power Draw | | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | Power Draw Limit | | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Clock Speeds | Current | Graphics | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | SM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | Memory | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Max | Graphics | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | SM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | Memory | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Utilization Rates | GPU Compute | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | PCIE Memory | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ECC Errors | Volatile | Location-Based | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | | Total | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | Aggregate | Location-Based | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| | | Total | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Performance State | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Process Info | Process Id | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Process Name | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Allocated Device Memory | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

| Commands | 4000 | 5000 | 6000 | Quadro Plex 7000 | M2070-Q | 600 | 2000 | 3000M | 410 | S1070 | C1060 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Set Compute Mode | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Set Persistence Mode (Linux-Only) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Set Display Model (Win7-Only) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Set ECC Mode | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Clear ECC Errors | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Reset GPU (Linux-Only) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Figure 1.3: This chart shows which features are available for each Quadro and T10 GPU product.

# Chapter 2

# Change log of NVML library

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

## 2.1 Changes between NVML v1.0 and v2.285

- 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 Event Handling Methods
- Added API nvmlDeviceGetComputeRunningProcesses and nvmlSystemGetProcessName functions for looking up currently running compute applications
- Deprecated nvmlDeviceGetPowerState in favor of nvmlDeviceGetPerformanceState.

## 2.2 Changes between NVML v2.285 and v3.295

- deprecated nvmlDeviceGetHandleBySerial in favor of newly added nvmlDeviceGetHandleByUUID
- Marked the input parameters of nvmlDeviceGetHandleBySerial, nvmlDeviceGetHandleByUUID and nvmlDeviceGetHandleByPciBusId as const
- Added nvmlDeviceOnSameBoard
- Added Constants defines
- Added nvmlDeviceGetMaxPcieLinkGeneration, nvmlDeviceGetMaxPcieLinkWidth, nvmlDeviceGetCurrPcieLinkGeneration,nvmlDeviceGetCurrPcieLinkWidth

- Format change of nvmlDeviceGetUUID output to match the UUID standard. This function will return a different value.
- nvmlDeviceGetDetailedEccErrors will report zero for unsupported ECC error counters when a subset of ECC error counters are supported

# Chapter 3

# Deprecated List

**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.

# Chapter 4

# Module Index

## 4.1 Modules

Here is a list of all modules:

# Chapter 5

# Data Structure Index

## 5.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 6

# Module Documentation

## 6.1 Device Structs

### Data Structures

- struct nvmlPciInfo_t
- struct nvmlEccErrorCounts_t
- struct nvmlUtilization_t
- struct nvmlMemory_t
- struct nvmlProcessInfo_t

### Defines

- #define NVML_VALUE_NOT_AVAILABLE (-1)

### 6.1.1 Define Documentation

#### 6.1.1.1 #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.

## 6.2 Device Enums

### Defines

- #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.*

### Enumerations

- enum nvmlEnableState_t {
  NVML_FEATURE_DISABLED = 0,
  NVML_FEATURE_ENABLED = 1 }
- enum nvmlTemperatureSensors_t { NVML_TEMPERATURE_GPU = 0 }
- enum nvmlComputeMode_t {
  NVML_COMPUTEMODE_DEFAULT = 0,
  NVML_COMPUTEMODE_EXCLUSIVE_THREAD = 1,
  NVML_COMPUTEMODE_PROHIBITED = 2,
  NVML_COMPUTEMODE_EXCLUSIVE_PROCESS = 3 }
- enum nvmlEccBitType_t {
  NVML_SINGLE_BIT_ECC = 0,
  NVML_DOUBLE_BIT_ECC = 1 }
- enum nvmlEccCounterType_t {
  NVML_VOLATILE_ECC = 0,
  NVML_AGGREGATE_ECC = 1 }
- enum nvmlClockType_t {
  NVML_CLOCK_GRAPHICS = 0,
  NVML_CLOCK_SM = 1,
  NVML_CLOCK_MEM = 2 }
- enum nvmlDriverModel_t {
  NVML_DRIVER_WDDM = 0,
  NVML_DRIVER_WDM = 1 }
- enum nvmlPstates_t {
  NVML_PSTATE_0 = 0,
  NVML_PSTATE_1 = 1,
  NVML_PSTATE_2 = 2,
  NVML_PSTATE_3 = 3,
  NVML_PSTATE_4 = 4,
  NVML_PSTATE_5 = 5,
  NVML_PSTATE_6 = 6,
  NVML_PSTATE_7 = 7,

NVML_PSTATE_8 = 8,
NVML_PSTATE_9 = 9,
NVML_PSTATE_10 = 10,
NVML_PSTATE_11 = 11,
NVML_PSTATE_12 = 12,
NVML_PSTATE_13 = 13,
NVML_PSTATE_14 = 14,
NVML_PSTATE_15 = 15,
NVML_PSTATE_UNKNOWN = 32 }

- enum nvmlInforomObject_t {
  NVML_INFOROM_OEM = 0,
  NVML_INFOROM_ECC = 1,
  NVML_INFOROM_POWER = 2 }
- enum nvmlReturn_t {
  NVML_SUCCESS = 0,
  NVML_ERROR_UNINITIALIZED = 1,
  NVML_ERROR_INVALID_ARGUMENT = 2,
  NVML_ERROR_NOT_SUPPORTED = 3,
  NVML_ERROR_NO_PERMISSION = 4,
  NVML_ERROR_ALREADY_INITIALIZED = 5,
  NVML_ERROR_NOT_FOUND = 6,
  NVML_ERROR_INSUFFICIENT_SIZE = 7,
  NVML_ERROR_INSUFFICIENT_POWER = 8,
  NVML_ERROR_DRIVER_NOT_LOADED = 9,
  NVML_ERROR_TIMEOUT = 10,
  NVML_ERROR_UNKNOWN = 999 }

## 6.2.1 Enumeration Type Documentation

### 6.2.1.1 enum nvmlClockType_t

Clock types.

All speeds are in Mhz.

**Enumerator:**

***NVML_CLOCK_GRAPHICS*** Graphics clock domain.

***NVML_CLOCK_SM*** SM clock domain.

***NVML_CLOCK_MEM*** Memory clock domain.

#### 6.2.1.2 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.

**Enumerator:**

***NVML_COMPUTEMODE_DEFAULT*** Default compute mode – multiple contexts per device.

***NVML_COMPUTEMODE_EXCLUSIVE_THREAD*** Compute-exclusive-thread mode – only one context per device, usable from one thread at a time.

***NVML_COMPUTEMODE_PROHIBITED*** Compute-prohibited mode – no contexts per device.

***NVML_COMPUTEMODE_EXCLUSIVE_PROCESS*** Compute-exclusive-process mode – only one context per device, usable from multiple threads at a time.

#### 6.2.1.3 enum nvmlDriverModel_t

Driver models.

Windows only.

**Enumerator:**

***NVML_DRIVER_WDDM*** WDDM driver model – GPU treated as a display device.

***NVML_DRIVER_WDM*** WDM (TCC) model (recommended) – GPU treated as a generic device.

#### 6.2.1.4 enum nvmlEccBitType_t

ECC bit types.

**Enumerator:**

***NVML_SINGLE_BIT_ECC*** Single bit ECC errors.

***NVML_DOUBLE_BIT_ECC*** Double bit ECC errors.

#### 6.2.1.5 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.

**Enumerator:**

***NVML_VOLATILE_ECC*** Volatile counts are reset each time the driver loads.

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

#### 6.2.1.6 enum nvmlEnableState_t

Generic enable/disable enum.

**Enumerator:**

***NVML_FEATURE_DISABLED*** Feature disabled.

***NVML_FEATURE_ENABLED*** Feature enabled.

#### 6.2.1.7 enum nvmlInforomObject_t

Available infoROM objects.

**Enumerator:**

***NVML_INFOROM_OEM*** An object defined by OEM.

***NVML_INFOROM_ECC*** The ECC object determining the level of ECC support.

***NVML_INFOROM_POWER*** The power management object.

#### 6.2.1.8 enum nvmlPstates_t

Allowed PStates.

**Enumerator:**

***NVML_PSTATE_0*** Performance state 0 – Maximum Performance.

***NVML_PSTATE_1*** Performance state 1.

***NVML_PSTATE_2*** Performance state 2.

***NVML_PSTATE_3*** Performance state 3.

***NVML_PSTATE_4*** Performance state 4.

***NVML_PSTATE_5*** Performance state 5.

***NVML_PSTATE_6*** Performance state 6.

***NVML_PSTATE_7*** Performance state 7.

***NVML_PSTATE_8*** Performance state 8.

***NVML_PSTATE_9*** Performance state 9.

***NVML_PSTATE_10*** Performance state 10.

***NVML_PSTATE_11*** Performance state 11.

***NVML_PSTATE_12*** Performance state 12.

***NVML_PSTATE_13*** Performance state 13.

***NVML_PSTATE_14*** Performance state 14.

***NVML_PSTATE_15*** Performance state 15 – Minimum Performance.

***NVML_PSTATE_UNKNOWN*** Unknown performance state.

#### 6.2.1.9 enum nvmlReturn_t

Return values for NVML API calls.

**Enumerator:**

***NVML_SUCCESS*** The operation was successful.

***NVML_ERROR_UNINITIALIZED*** NVML was not first initialized with nvmlInit().

***NVML_ERROR_INVALID_ARGUMENT*** A supplied argument is invalid.

***NVML_ERROR_NOT_SUPPORTED*** The requested operation is not available on target device.

***NVML_ERROR_NO_PERMISSION*** The current user does not have permission for operation.

***NVML_ERROR_ALREADY_INITIALIZED*** Deprecated: Multiple initializations are now allowed through ref counting.

***NVML_ERROR_NOT_FOUND*** A query to find an object was unsuccessful.

***NVML_ERROR_INSUFFICIENT_SIZE*** An input argument is not large enough.

***NVML_ERROR_INSUFFICIENT_POWER*** A device's external power cables are not properly attached.

***NVML_ERROR_DRIVER_NOT_LOADED*** NVIDIA driver is not loaded.

***NVML_ERROR_TIMEOUT*** User provided timeout passed.

***NVML_ERROR_UNKNOWN*** An internal driver error occurred.

#### 6.2.1.10 enum nvmlTemperatureSensors_t

Temperature sensors.

**Enumerator:**

***NVML_TEMPERATURE_GPU*** Temperature sensor for the GPU die.

# 6.3 Unit Structs

## Data Structures

- struct nvmlHwbcEntry_t
- struct nvmlLedState_t
- struct nvmlUnitInfo_t
- struct nvmlPSUInfo_t
- struct nvmlUnitFanInfo_t
- struct nvmlUnitFanSpeeds_t

## Enumerations

- enum nvmlFanState_t {
  NVML_FAN_NORMAL = 0,
  NVML_FAN_FAILED = 1 }
- enum nvmlLedColor_t {
  NVML_LED_COLOR_GREEN = 0,
  NVML_LED_COLOR_AMBER = 1 }

### 6.3.1 Enumeration Type Documentation

#### 6.3.1.1 enum nvmlFanState_t

Fan state enum.

**Enumerator:**

***NVML_FAN_NORMAL*** Fan is working properly.

***NVML_FAN_FAILED*** Fan has failed.

#### 6.3.1.2 enum nvmlLedColor_t

Led color enum.

**Enumerator:**

***NVML_LED_COLOR_GREEN*** GREEN, indicates good health.

***NVML_LED_COLOR_AMBER*** AMBER, indicates problem.

## 6.4 Event Types

### Defines

- #define nvmlEventTypeSingleBitEccError 0x0000000000000001LL

  *Event about single bit ECC errors.*

- #define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL

  *Event about double bit ECC errors.*

- #define nvmlEventTypePState 0x0000000000000004LL

  *Event about PState changes.*

- #define nvmlEventTypeXidCriticalError 0x0000000000000008LL

  *Event that Xid critical error occurred.*

- #define nvmlEventTypeNone 0x0000000000000000LL

  *Mask with no events.*

- #define nvmlEventTypeAll

  *Mask of all events.*

### 6.4.1 Detailed Description

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

### 6.4.2 Define Documentation

#### 6.4.2.1 #define nvmlEventTypePState 0x0000000000000004LL

**Note:**

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.

# 6.5 Initialization and Cleanup

## Functions

- nvmlReturn_t DECLDIR nvmlInit (void)
- nvmlReturn_t DECLDIR nvmlShutdown (void)

## 6.5.1 Detailed Description

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.

## 6.5.2 Function Documentation

### 6.5.2.1 nvmlReturn_t DECLDIR nvmlInit (void)

Initialize NVML by discovering and attaching to all GPU devices in the system.

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.

**Returns:**

- NVML_SUCCESS if NVML has been properly initialized
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to talk to any device
- NVML_ERROR_DRIVER_NOT_LOADED if NVIDIA driver is not running
- NVML_ERROR_INSUFFICIENT_POWER if any devices have improperly attached external power cables
- NVML_ERROR_UNKNOWN on any unexpected error

### 6.5.2.2 nvmlReturn_t DECLDIR nvmlShutdown (void)

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().

**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

# 6.6 Error reporting

## Functions

- const DECLDIR char ∗ nvmlErrorString (nvmlReturn_t result)

### 6.6.1 Detailed Description

This chapter describes helper functions for error reporting routines.

### 6.6.2 Function Documentation

#### 6.6.2.1 const DECLDIR char∗ nvmlErrorString (nvmlReturn_t *result*)

Helper method for converting NVML error codes into readable strings.

For all products

**Parameters:**

***result*** NVML error code to convert

**Returns:**

String representation of the error.

# 6.7 Constants

## Defines

- #define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16
- #define NVML_DEVICE_UUID_BUFFER_SIZE 80
- #define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80
- #define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80
- #define NVML_DEVICE_NAME_BUFFER_SIZE 64
- #define NVML_DEVICE_SERIAL_BUFFER_SIZE 30
- #define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32

## 6.7.1 Define Documentation

### 6.7.1.1 #define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16

Buffer size guaranteed to be large enough for nvmlDeviceGetInforomVersion

### 6.7.1.2 #define NVML_DEVICE_NAME_BUFFER_SIZE 64

Buffer size guaranteed to be large enough for nvmlDeviceGetName

### 6.7.1.3 #define NVML_DEVICE_SERIAL_BUFFER_SIZE 30

Buffer size guaranteed to be large enough for nvmlDeviceGetSerial

### 6.7.1.4 #define NVML_DEVICE_UUID_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlDeviceGetUUID

### 6.7.1.5 #define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32

Buffer size guaranteed to be large enough for nvmlDeviceGetVbiosVersion

### 6.7.1.6 #define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlSystemGetDriverVersion

### 6.7.1.7 #define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlSystemGetNVMLVersion

## 6.8 System Queries

### Functions

- nvmlReturn_t DECLDIR nvmlSystemGetDriverVersion (char ∗version, unsigned int length)
- nvmlReturn_t DECLDIR nvmlSystemGetNVMLVersion (char ∗version, unsigned int length)
- nvmlReturn_t DECLDIR nvmlSystemGetProcessName (unsigned int pid, char ∗name, unsigned int length)

### 6.8.1 Detailed Description

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

### 6.8.2 Function Documentation

#### 6.8.2.1 nvmlReturn_t DECLDIR nvmlSystemGetDriverVersion (char ∗ *version*, unsigned int *length*)

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.

**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

#### 6.8.2.2 nvmlReturn_t DECLDIR nvmlSystemGetNVMLVersion (char ∗ *version*, unsigned int *length*)

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.

**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

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

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.

**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
- 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

## 6.9 Unit Queries

### Functions

- nvmlReturn_t DECLDIR nvmlUnitGetCount (unsigned int *unitCount)
- nvmlReturn_t DECLDIR nvmlUnitGetHandleByIndex (unsigned int index, nvmlUnit_t *unit)
- nvmlReturn_t DECLDIR nvmlUnitGetUnitInfo (nvmlUnit_t unit, nvmlUnitInfo_t *info)
- nvmlReturn_t DECLDIR nvmlUnitGetLedState (nvmlUnit_t unit, nvmlLedState_t *state)
- nvmlReturn_t DECLDIR nvmlUnitGetPsuInfo (nvmlUnit_t unit, nvmlPSUInfo_t *psu)
- nvmlReturn_t DECLDIR nvmlUnitGetTemperature (nvmlUnit_t unit, unsigned int type, unsigned int *temp)
- nvmlReturn_t DECLDIR nvmlUnitGetFanSpeedInfo (nvmlUnit_t unit, nvmlUnitFanSpeeds_t *fanSpeeds)
- nvmlReturn_t DECLDIR nvmlUnitGetDevices (nvmlUnit_t unit, unsigned int *deviceCount, nvmlDevice_t *devices)
- nvmlReturn_t DECLDIR nvmlSystemGetHicVersion (unsigned int *hwbcCount, nvmlHwbcEntry_t *hwbcEntries)

### 6.9.1 Detailed Description

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().

### 6.9.2 Function Documentation

#### 6.9.2.1 nvmlReturn_t DECLDIR nvmlSystemGetHicVersion (unsigned int * *hwbcCount*, nvmlHwbcEntry_t * *hwbcEntries*)

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.

**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

#### 6.9.2.2 nvmlReturn_t DECLDIR nvmlUnitGetCount (unsigned int * *unitCount*)

Retrieves the number of units in the system.

For S-class products.

**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

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

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.

**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

#### 6.9.2.4 nvmlReturn_t DECLDIR nvmlUnitGetFanSpeedInfo (nvmlUnit_t *unit*, nvmlUnitFanSpeeds_t * *fanSpeeds*)

Retrieves the fan speed readings for the unit.

For S-class products.

See nvmlUnitFanSpeeds_t for details on available fan speed info.

**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

#### 6.9.2.5 nvmlReturn_t DECLDIR nvmlUnitGetHandleByIndex (unsigned int *index*, nvmlUnit_t * *unit*)

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.

**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

#### 6.9.2.6 nvmlReturn_t DECLDIR nvmlUnitGetLedState (nvmlUnit_t *unit*, nvmlLedState_t * *state*)

Retrieves the LED state associated with this unit.

For S-class products.

See nvmlLedState_t for details on allowed states.

**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

**See also:**

nvmlUnitSetLedState()

#### 6.9.2.7 nvmlReturn_t DECLDIR nvmlUnitGetPsuInfo (nvmlUnit_t *unit*, nvmlPSUInfo_t * *psu*)

Retrieves the PSU stats for the unit.

For S-class products.

See nvmlPSUInfo_t for details on available PSU info.

**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

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

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).

**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

#### 6.9.2.9 nvmlReturn_t DECLDIR nvmlUnitGetUnitInfo (nvmlUnit_t *unit*, nvmlUnitInfo_t * *info*)

Retrieves the static information associated with a unit.

For S-class products.

See nvmlUnitInfo_t for details on available unit 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

## 6.10 Device Queries

### Functions

- nvmlReturn_t DECLDIR nvmlDeviceGetCount (unsigned int *deviceCount)
- nvmlReturn_t DECLDIR nvmlDeviceGetHandleByIndex (unsigned int index, nvmlDevice_t *device)
- nvmlReturn_t DECLDIR nvmlDeviceGetHandleBySerial (const char *serial, nvmlDevice_t *device)
- nvmlReturn_t DECLDIR nvmlDeviceGetHandleByUUID (const char *uuid, nvmlDevice_t *device)
- nvmlReturn_t DECLDIR nvmlDeviceGetHandleByPciBusId (const char *pciBusId, nvmlDevice_t *device)
- nvmlReturn_t DECLDIR nvmlDeviceGetName (nvmlDevice_t device, char *name, unsigned int length)
- nvmlReturn_t DECLDIR nvmlDeviceGetSerial (nvmlDevice_t device, char *serial, unsigned int length)
- nvmlReturn_t DECLDIR nvmlDeviceGetUUID (nvmlDevice_t device, char *uuid, unsigned int length)
- nvmlReturn_t DECLDIR nvmlDeviceGetInforomVersion (nvmlDevice_t device, nvmlInforomObject_t object, char *version, unsigned int length)
- nvmlReturn_t DECLDIR nvmlDeviceGetDisplayMode (nvmlDevice_t device, nvmlEnableState_t *display)
- nvmlReturn_t DECLDIR nvmlDeviceGetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t *mode)
- nvmlReturn_t DECLDIR nvmlDeviceGetPciInfo (nvmlDevice_t device, nvmlPciInfo_t *pci)
- nvmlReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkGeneration (nvmlDevice_t device, unsigned int *maxLinkGen)
- nvmlReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkWidth (nvmlDevice_t device, unsigned int *maxLinkWidth)
- nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkGeneration (nvmlDevice_t device, unsigned int *currLinkGen)
- nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkWidth (nvmlDevice_t device, unsigned int *currLinkWidth)
- nvmlReturn_t DECLDIR nvmlDeviceGetClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)
- nvmlReturn_t DECLDIR nvmlDeviceGetMaxClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)
- nvmlReturn_t DECLDIR nvmlDeviceGetFanSpeed (nvmlDevice_t device, unsigned int *speed)
- nvmlReturn_t DECLDIR nvmlDeviceGetTemperature (nvmlDevice_t device, nvmlTemperatureSensors_t sensorType, unsigned int *temp)
- nvmlReturn_t DECLDIR nvmlDeviceGetPerformanceState (nvmlDevice_t device, nvmlPstates_t *pState)
- nvmlReturn_t DECLDIR nvmlDeviceGetPowerState (nvmlDevice_t device, nvmlPstates_t *pState)
- nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementMode (nvmlDevice_t device, nvmlEnableState_t *mode)
- nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementLimit (nvmlDevice_t device, unsigned int *limit)
- nvmlReturn_t DECLDIR nvmlDeviceGetPowerUsage (nvmlDevice_t device, unsigned int *power)
- nvmlReturn_t DECLDIR nvmlDeviceGetMemoryInfo (nvmlDevice_t device, nvmlMemory_t *memory)
- nvmlReturn_t DECLDIR nvmlDeviceGetComputeMode (nvmlDevice_t device, nvmlComputeMode_t *mode)
- nvmlReturn_t DECLDIR nvmlDeviceGetEccMode (nvmlDevice_t device, nvmlEnableState_t *current, nvmlEnableState_t *pending)
- nvmlReturn_t DECLDIR nvmlDeviceGetTotalEccErrors (nvmlDevice_t device, nvmlEccBitType_t bitType, nvmlEccCounterType_t counterType, unsigned long long *eccCounts)
- nvmlReturn_t DECLDIR nvmlDeviceGetDetailedEccErrors (nvmlDevice_t device, nvmlEccBitType_t bitType, nvmlEccCounterType_t counterType, nvmlEccErrorCounts_t *eccCounts)
- nvmlReturn_t DECLDIR nvmlDeviceGetUtilizationRates (nvmlDevice_t device, nvmlUtilization_t *utilization)
- nvmlReturn_t DECLDIR nvmlDeviceGetDriverModel (nvmlDevice_t device, nvmlDriverModel_t *current, nvmlDriverModel_t *pending)
- nvmlReturn_t DECLDIR nvmlDeviceGetVbiosVersion (nvmlDevice_t device, char *version, unsigned int length)

- nvmlReturn_t DECLDIR nvmlDeviceGetComputeRunningProcesses (nvmlDevice_t device, unsigned int ∗infoCount, nvmlProcessInfo_t ∗infos)
- nvmlReturn_t DECLDIR nvmlDeviceOnSameBoard (nvmlDevice_t device1, nvmlDevice_t device2, int ∗onSameBoard)

### 6.10.1 Detailed Description

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() or nvmlDeviceGetHandleByPciBusId().

### 6.10.2 Function Documentation

#### 6.10.2.1 nvmlReturn_t DECLDIR nvmlDeviceGetClockInfo (nvmlDevice_t *device*, nvmlClockType_t *type*, unsigned int ∗ *clock*)

Retrieves the current clock speeds for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

See nvmlClockType_t for details on available clock information.

**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_UNKNOWN on any unexpected error

#### 6.10.2.2 nvmlReturn_t DECLDIR nvmlDeviceGetComputeMode (nvmlDevice_t *device*, nvmlComputeMode_t ∗ *mode*)

Retrieves the current compute mode for the device.

For all CUDA-capable products.

See nvmlComputeMode_t for details on allowed compute modes.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceSetComputeMode()

#### 6.10.2.3 nvmlReturn_t DECLDIR nvmlDeviceGetComputeRunningProcesses (nvmlDevice_t *device*, unsigned int * *infoCount*, nvmlProcessInfo_t * *infos*)

Get information about processes with a compute context on a device

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

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.

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.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlSystemGetProcessName

#### 6.10.2.4 nvmlReturn_t DECLDIR nvmlDeviceGetCount (unsigned int * *deviceCount*)

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

For all products.

On some platforms not all devices may be accessible due to permission restrictions. In these cases the device count will reflect only the GPUs that NVML can access.

**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

#### 6.10.2.5 nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkGeneration (nvmlDevice_t *device*, unsigned int * *currLinkGen*)

Retrieves the current PCIe link generation

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

**Parameters:**

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

***currLinkGen*** Reference in which to return the max 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_UNKNOWN on any unexpected error

#### 6.10.2.6 nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkWidth (nvmlDevice_t *device*, unsigned int * *currLinkWidth*)

Retrieves the current PCIe link width

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

**Parameters:**

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

***currLinkWidth*** Reference in which to return the max 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_UNKNOWN on any unexpected error

#### 6.10.2.7 nvmlReturn_t DECLDIR nvmlDeviceGetDetailedEccErrors (nvmlDevice_t *device*, nvmlEccBitType_t *bitType*, nvmlEccCounterType_t *counterType*, nvmlEccErrorCounts_t * *eccCounts*)

Retrieves the detailed ECC error counts for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families. 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 nvmlEccBitType_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.

**Parameters:**

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

***bitType*** Flag that specifies the bit-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*, *bitType* or *counterType* is invalid, or *eccCounts* is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceClearEccErrorCounts()

#### 6.10.2.8 nvmlReturn_t DECLDIR nvmlDeviceGetDisplayMode (nvmlDevice_t *device*, nvmlEnableState_t * *display*)

Retrieves the display mode for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

This method indicates whether a physical display is currently connected to the device.

See nvmlEnableState_t for details on allowed modes.

**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

#### 6.10.2.9 nvmlReturn_t DECLDIR nvmlDeviceGetDriverModel (nvmlDevice_t *device*, nvmlDriverModel_t * *current*, nvmlDriverModel_t * *pending*)

Retrieves the current and pending driver model for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families. 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.

**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 *current* and *pending* have been populated
- 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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceSetDriverModel()

#### 6.10.2.10 nvmlReturn_t DECLDIR nvmlDeviceGetEccMode (nvmlDevice_t *device*, nvmlEnableState_t * *current*, nvmlEnableState_t * *pending*)

Retrieves the current and pending ECC modes for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families. 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.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceSetEccMode()

#### 6.10.2.11 nvmlReturn_t DECLDIR nvmlDeviceGetFanSpeed (nvmlDevice_t *device*, unsigned int * *speed*)

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

For all discrete products with dedicated fans.

The fan speed is expressed as a percent of the maximum, i.e. full speed is 100%.

**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_UNKNOWN on any unexpected error

#### 6.10.2.12 nvmlReturn_t DECLDIR nvmlDeviceGetHandleByIndex (unsigned int *index*, nvmlDevice_t * *device*)

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 board serial numbers. See nvmlDeviceGetHandleBySerial() and nvmlDeviceGetHandleByPciBusId().

**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_UNKNOWN on any unexpected error

#### 6.10.2.13 nvmlReturn_t DECLDIR nvmlDeviceGetHandleByPciBusId (const char * *pciBusId*, nvmlDevice_t * *device*)

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().

**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_UNKNOWN on any unexpected error

#### 6.10.2.14 nvmlReturn_t DECLDIR nvmlDeviceGetHandleBySerial (const char * *serial*, nvmlDevice_t * *device*)

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

For all products.

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.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceGetSerial
nvmlDeviceGetHandleByUUID

#### 6.10.2.15 nvmlReturn_t DECLDIR nvmlDeviceGetHandleByUUID (const char * *uuid*, nvmlDevice_t * *device*)

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

For all products.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceGetUUID

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

Retrieves the version information for the device's infoROM.

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

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.

**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_UNKNOWN on any unexpected error

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

Retrieves the maximum clock speeds for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

See nvmlClockType_t for details on available clock information.

**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_UNKNOWN on any unexpected error

#### 6.10.2.18 nvmlReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkGeneration (nvmlDevice_t *device*, unsigned int * *maxLinkGen*)

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 Tesla ™and Quadro ®products from the Fermi and Kepler families.

**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_UNKNOWN on any unexpected error

#### 6.10.2.19 nvmlReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkWidth (nvmlDevice_t *device*, unsigned int * *maxLinkWidth*)

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 Tesla ™and Quadro ®products from the Fermi and Kepler families.

**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_UNKNOWN on any unexpected error

#### 6.10.2.20 nvmlReturn_t DECLDIR nvmlDeviceGetMemoryInfo (nvmlDevice_t *device*, nvmlMemory_t * *memory*)

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.

**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_UNKNOWN on any unexpected error

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

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.

**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

#### 6.10.2.22 nvmlReturn_t DECLDIR nvmlDeviceGetPciInfo (nvmlDevice_t *device*, nvmlPciInfo_t * *pci*)

Retrieves the PCI attributes of this device.

For all products.

See nvmlPciInfo_t for details on the available PCI info.

**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_UNKNOWN on any unexpected error

#### 6.10.2.23 nvmlReturn_t DECLDIR nvmlDeviceGetPerformanceState (nvmlDevice_t *device*, nvmlPstates_t * *pState*)

Retrieves the current performance state for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

See nvmlPstates_t for details on allowed performance states.

**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_UNKNOWN on any unexpected error

#### 6.10.2.24 nvmlReturn_t DECLDIR nvmlDeviceGetPersistenceMode (nvmlDevice_t *device*, nvmlEnableState_t * *mode*)

Retrieves the persistence mode associated with this device.

For all CUDA-capable 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.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceSetPersistenceMode()

#### 6.10.2.25 nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementLimit (nvmlDevice_t *device*, unsigned int * *limit*)

Retrieves the power management limit associated with this device, in milliwatts.

For "GF11x" Tesla ™and Quadro ®products from the Fermi family.

- Requires *NVML_INFOROM_POWER* version 3.0 or higher.

For Tesla ™and Quadro ®products from the Kepler family.

- Does not require *NVML_INFOROM_POWER* object.

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.

**Parameters:**

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

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

**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_UNKNOWN on any unexpected error

#### 6.10.2.26 nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementMode (nvmlDevice_t *device*, nvmlEnableState_t * *mode*)

Retrieves the power management mode associated with this device.

For "GF11x" Tesla ™and Quadro ®products from the Fermi family.

- Requires *NVML_INFOROM_POWER* version 3.0 or higher.

For Tesla ™and Quadro ®products from the Kepler family.

- Does not require *NVML_INFOROM_POWER* object.

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

See nvmlEnableState_t for details on allowed modes.

**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_UNKNOWN on any unexpected error

#### 6.10.2.27 nvmlReturn_t DECLDIR nvmlDeviceGetPowerState (nvmlDevice_t *device*, nvmlPstates_t * *pState*)

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

Retrieve the current power state for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

See nvmlPstates_t for details on allowed power states.

**Parameters:**

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

***pState*** Reference in which to return the power 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_UNKNOWN on any unexpected error

#### 6.10.2.28 nvmlReturn_t DECLDIR nvmlDeviceGetPowerUsage (nvmlDevice_t *device*, unsigned int * *power*)

Retrieves the power usage reading for the device, in milliwatts. This is the power draw for the entire board, including GPU, memory, etc.

For "GF11x" Tesla ™and Quadro ®products from the Fermi family.

- Requires *NVML_INFOROM_POWER* version 3.0 or higher.

For Tesla ™and Quadro ®products from the Kepler family.

- Does not require *NVML_INFOROM_POWER* object.

The reading is accurate to within a range of +/- 5 watts. It is only available if power management mode is supported. See nvmlDeviceGetPowerManagementMode.

**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_UNKNOWN on any unexpected error

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

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

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

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.

**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

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

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

For all discrete and S-class products.

See nvmlTemperatureSensors_t for details on available temperature sensors.

**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_UNKNOWN on any unexpected error

#### 6.10.2.31 nvmlReturn_t DECLDIR nvmlDeviceGetTotalEccErrors (nvmlDevice_t *device*, nvmlEccBitType_t *bitType*, nvmlEccCounterType_t *counterType*, unsigned long long * *eccCounts*)

Retrieves the total ECC error counts for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families. 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 nvmlEccBitType_t for a description of available bit types.

See nvmlEccCounterType_t for a description of available counter types.

**Parameters:**

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

***bitType*** Flag that specifies the bit-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*, *bitType* or *counterType* is invalid, or *eccCounts* is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceClearEccErrorCounts()

#### 6.10.2.32 nvmlReturn_t DECLDIR nvmlDeviceGetUtilizationRates (nvmlDevice_t *device*, nvmlUtilization_t * *utilization*)

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

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

See nvmlUtilization_t for details on available utilization rates.

**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_UNKNOWN on any unexpected error

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

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

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

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 nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE.

**Parameters:**

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

***uuid*** Reference in which to return the GPU UUID

***length*** The maximum allowed length of the string returned in *uuid*

**Returns:**

- NVML_SUCCESS if *uuid* has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if *device* is invalid, or *uuid* is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if *length* is too small
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

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

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.

**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_UNKNOWN on any unexpected error

#### 6.10.2.35 nvmlReturn_t DECLDIR nvmlDeviceOnSameBoard (nvmlDevice_t *device1*, nvmlDevice_t *device2*, int * *onSameBoard*)

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

**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 when onSameBoard has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if *dev1*, *dev2* or *onSameBoard* are invalid
- NVML_ERROR_UNKNOWN on any unexpected error

# 6.11 Unit Commands

## Functions

- nvmlReturn_t DECLDIR nvmlUnitSetLedState (nvmlUnit_t unit, nvmlLedColor_t color)

## 6.11.1 Detailed Description

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.

## 6.11.2 Function Documentation

### 6.11.2.1 nvmlReturn_t DECLDIR nvmlUnitSetLedState (nvmlUnit_t *unit*, nvmlLedColor_t *color*)

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.

**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

**See also:**

nvmlUnitGetLedState()

## 6.12 Device Commands

### Functions

- nvmlReturn_t DECLDIR nvmlDeviceSetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t mode)
- nvmlReturn_t DECLDIR nvmlDeviceSetComputeMode (nvmlDevice_t device, nvmlComputeMode_t mode)
- nvmlReturn_t DECLDIR nvmlDeviceSetEccMode (nvmlDevice_t device, nvmlEnableState_t ecc)
- nvmlReturn_t DECLDIR nvmlDeviceClearEccErrorCounts (nvmlDevice_t device, nvmlEccCounterType_t counterType)
- nvmlReturn_t DECLDIR nvmlDeviceSetDriverModel (nvmlDevice_t device, nvmlDriverModel_t driverModel, unsigned int flags)

### 6.12.1 Detailed Description

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.

### 6.12.2 Function Documentation

#### 6.12.2.1 nvmlReturn_t DECLDIR nvmlDeviceClearEccErrorCounts (nvmlDevice_t *device*, nvmlEccCounterType_t *counterType*)

Clear the ECC error counts for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families. 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 nvmlEccCounterType_t for details on available counter types.

**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_UNKNOWN on any unexpected error

**See also:**

- nvmlDeviceGetDetailedEccErrors()
- nvmlDeviceGetTotalEccErrors()

#### 6.12.2.2 nvmlReturn_t DECLDIR nvmlDeviceSetComputeMode (nvmlDevice_t *device*, nvmlComputeMode_t *mode*)

Set the compute mode for the device.

For all CUDA-capable 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 for details on available compute modes.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceGetComputeMode()

#### 6.12.2.3 nvmlReturn_t DECLDIR nvmlDeviceSetDriverModel (nvmlDevice_t *device*, nvmlDriverModel_t *driverModel*, unsigned int *flags*)

Set the driver model for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families. 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.

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

See nvmlDriverModel_t for details on available driver models. See nvmlFlagDefault and nvmlFlagForce

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceGetDriverModel()

#### 6.12.2.4 nvmlReturn_t DECLDIR nvmlDeviceSetEccMode (nvmlDevice_t *device*, nvmlEnableState_t *ecc*)

Set the ECC mode for the device.

For Tesla ™and Quadro ®products from the Fermi and Kepler families. 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.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceGetEccMode()

#### 6.12.2.5 nvmlReturn_t DECLDIR nvmlDeviceSetPersistenceMode (nvmlDevice_t *device*, nvmlEnableState_t *mode*)

Set the persistence mode for the device.

For all CUDA-capable 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.

**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_UNKNOWN on any unexpected error

**See also:**

nvmlDeviceGetPersistenceMode()

## 6.13 Event Handling Methods

### Data Structures

- struct nvmlEventData_t

### Modules

- Event Types

### Typedefs

- typedef struct nvmlEventSet_st * nvmlEventSet_t

### Functions

- nvmlReturn_t DECLDIR nvmlEventSetCreate (nvmlEventSet_t *set)
- nvmlReturn_t DECLDIR nvmlDeviceRegisterEvents (nvmlDevice_t device, unsigned long long eventTypes, nvmlEventSet_t set)
- nvmlReturn_t DECLDIR nvmlDeviceGetSupportedEventTypes (nvmlDevice_t device, unsigned long long *eventTypes)
- nvmlReturn_t DECLDIR nvmlEventSetWait (nvmlEventSet_t set, nvmlEventData_t *data, unsigned int timeoutms)
- nvmlReturn_t DECLDIR nvmlEventSetFree (nvmlEventSet_t set)

### 6.13.1 Detailed Description

This chapter describes methods that NVML can perform against each device to register and wait for some event to occur.

### 6.13.2 Typedef Documentation

#### 6.13.2.1 typedef struct nvmlEventSet_st* nvmlEventSet_t

Handle to an event set

### 6.13.3 Function Documentation

#### 6.13.3.1 nvmlReturn_t DECLDIR nvmlDeviceGetSupportedEventTypes (nvmlDevice_t *device*, unsigned long long * *eventTypes*)

Returns information about events supported on device

For all products.

Events are not supported on Windows. So this function returns an empty mask in *eventTypes* on Windows.

**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_UNKNOWN on any unexpected error

**See also:**

Event Types
nvmlDeviceRegisterEvents

#### 6.13.3.2 nvmlReturn_t DECLDIR nvmlDeviceRegisterEvents (nvmlDevice_t *device*, unsigned long long *eventTypes*, nvmlEventSet_t *set*)

Starts recording of events on a specified devices and add the events to specified nvmlEventSet_t

For Tesla ™and Quadro ®products from the Fermi and Kepler families. 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

**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_UNKNOWN on any unexpected error

**See also:**

Event Types
nvmlDeviceGetSupportedEventTypes
nvmlEventSetWait
nvmlEventSetFree

#### 6.13.3.3 nvmlReturn_t DECLDIR nvmlEventSetCreate (nvmlEventSet_t * *set*)

Create an empty set of events. Event set should be freed by nvmlEventSetFree

**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

**See also:**

nvmlEventSetFree

#### 6.13.3.4 nvmlReturn_t DECLDIR nvmlEventSetFree (nvmlEventSet_t *set*)

Releases events in the set

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

**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

**See also:**

nvmlDeviceRegisterEvents

#### 6.13.3.5 nvmlReturn_t DECLDIR nvmlEventSetWait (nvmlEventSet_t *set*, nvmlEventData_t * *data*, unsigned int *timeoutms*)

Waits on events and delivers events

For Tesla ™and Quadro ®products from the Fermi and Kepler families.

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)

**Parameters:**

***set*** Reference to set of events to wait on

***data*** Reference in which to return event data

***timeoutms*** Maximum amount of wait time in ms 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_UNKNOWN on any unexpected error

**See also:**

Event Types
nvmlDeviceRegisterEvents

# Chapter 7

# Data Structure Documentation

## 7.1 nvmlEccErrorCounts_t Struct Reference

`#include <nvml.h>`

### Data Fields

- unsigned long long l1Cache
  *L1 cache errors.*
- unsigned long long l2Cache
  *L2 cache errors.*
- unsigned long long deviceMemory
  *Device memory errors.*
- unsigned long long registerFile
  *Register file errors.*

### 7.1.1 Detailed Description

Detailed ECC error counts for a device.

## 7.2 nvmlEventData_t Struct Reference

`#include <nvml.h>`

### Data Fields

- nvmlDevice_t device

  *Specific device where the event occurred.*

- unsigned long long eventType

  *Information about what specific event occurred.*

### 7.2.1 Detailed Description

Information about occurred event

## 7.3 nvmlHwbcEntry_t Struct Reference

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

### 7.3.1 Detailed Description

Description of HWBC entry

## 7.4 nvmlLedState_t Struct Reference

`#include <nvml.h>`

### Data Fields

- char cause [256]

  *If amber, a text description of the cause.*

- nvmlLedColor_t color

  *GREEN or AMBER.*

### 7.4.1 Detailed Description

LED states for an S-class unit.

## 7.5 nvmlMemory_t Struct Reference

`#include <nvml.h>`

### Data Fields

- unsigned long long total

  *Total installed FB memory (in bytes).*

- unsigned long long free

  *Unallocated FB memory (in bytes).*

- unsigned long long used

  *Allocated FB memory (in bytes). Note that the driver/GPU always sets aside a small amount of memory for bookkeeping.*

### 7.5.1 Detailed Description

Memory allocation information for a device.

## 7.6 nvmlPciInfo_t Struct Reference

`#include <nvml.h>`

### Data Fields

- char busId [16]

  *The tuple domain:bus:device.function PCI identifier (& NULL terminator).*

- unsigned int domain

  *The PCI domain on which the device's bus resides, 0 to 0xffff.*

- unsigned int bus

  *The bus on which the device resides, 0 to 0xff.*

- unsigned int device

  *The device's id on the bus, 0 to 31.*

- unsigned int pciDeviceId

  *The combined 16-bit device id and 16-bit vendor id.*

- unsigned int pciSubSystemId

  *The 32-bit Sub System Device ID.*

### 7.6.1 Detailed Description

PCI information about a GPU device.

# 7.7 nvmlProcessInfo_t Struct Reference

`#include <nvml.h>`

## Data Fields

- unsigned int pid

  *Process ID.*

- unsigned long long usedGpuMemory

  *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.*

### 7.7.1 Detailed Description

Information about running compute processes on the GPU

## 7.8 nvmlPSUInfo_t Struct Reference

`#include <nvml.h>`

### Data Fields

- char state [256]
  *The power supply state.*
- unsigned int current
  *PSU current (A).*
- unsigned int voltage
  *PSU voltage (V).*
- unsigned int power
  *PSU power draw (W).*

### 7.8.1 Detailed Description

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

# 7.9 nvmlUnitFanInfo_t Struct Reference

`#include <nvml.h>`

## Data Fields

- unsigned int speed

  *Fan speed (RPM).*

- nvmlFanState_t state

  *Flag that indicates whether fan is working properly.*

### 7.9.1 Detailed Description

Fan speed reading for a single fan in an S-class unit.

## 7.10 nvmlUnitFanSpeeds_t Struct Reference

`#include <nvml.h>`

### Data Fields

- nvmlUnitFanInfo_t fans [24]

  *Fan speed data for each fan.*

- unsigned int count

  *Number of fans in unit.*

### 7.10.1 Detailed Description

Fan speed readings for an entire S-class unit.

## 7.11 nvmlUnitInfo_t Struct Reference

`#include <nvml.h>`

### Data Fields

- char name [96]
  *Product name.*
- char id [96]
  *Product identifier.*
- char serial [96]
  *Product serial number.*
- char firmwareVersion [96]
  *Firmware version.*

### 7.11.1 Detailed Description

Static S-class unit info.

## 7.12 nvmlUtilization_t Struct Reference

`#include <nvml.h>`

### Data Fields

- unsigned int gpu

  *Percent of time over the past second during which one or more kernels was executing on the GPU.*

- unsigned int memory

  *Percent of time over the past second during which global (device) memory was being read or written.*

### 7.12.1 Detailed Description

Utilization information for a device.

# Index

## Notice



## Trademarks



## Copyright