NVML API REFERENCE MANUAL

January 7, 2013

Version 4.304.75

Contents

1 NVML API Reference **1**

1.1 Feature Matrix 4

2 Known issues in the current version of NVML library **9**

3 Change log of NVML library **11**

3.1 Changes between NVML v4.304 RC and v4.304 Production 12

3.2 Changes between NVML v4.304 RC and v4.304 Production 12

3.3 Changes between NVML v2.285 and v3.295 12

3.4 Changes between NVML v1.0 and v2.285 13

4 Deprecated List **15**

5 Module Index **17**

5.1 Modules 17

6 Data Structure Index **19**

6.1 Data Structures 19

7 Module Documentation **21**

7.1 Device Structs 21

7.1.1 Define Documentation 21

7.1.1.1 NVML_VALUE_NOT_AVAILABLE 21

7.2 Device Enums 22

7.2.1 Define Documentation 24

7.2.1.1 NVML_DOUBLE_BIT_ECC 24

7.2.1.2 NVML_SINGLE_BIT_ECC 24

7.2.1.3 nvmlEccBitType_t 24

7.2.2 Enumeration Type Documentation 24

7.2.2.1 nvmlClockType_t 24

7.2.2.2 nvmlComputeMode_t 25

7.2.2.3 nvmlDriverModel_t . . . 25
7.2.2.4 nvmlEccCounterType_t . . . 25
7.2.2.5 nvmlEnableState_t . . . 25
7.2.2.6 nvmlGpuOperationMode_t . . . 26
7.2.2.7 nvmlInforomObject_t . . . 26
7.2.2.8 nvmlMemoryErrorType_t . . . 26
7.2.2.9 nvmlMemoryLocation_t . . . 26
7.2.2.10 nvmlPstates_t . . . 27
7.2.2.11 nvmlReturn_t . . . 27
7.2.2.12 nvmlTemperatureSensors_t . . . 28
7.3 Unit Structs . . . 29
7.3.1 Enumeration Type Documentation . . . 29
7.3.1.1 nvmlFanState_t . . . 29
7.3.1.2 nvmlLedColor_t . . . 29
7.4 Event Types . . . 30
7.4.1 Detailed Description . . . 30
7.4.2 Define Documentation . . . 30
7.4.2.1 nvmlEventTypeClock . . . 30
7.4.2.2 nvmlEventTypePState . . . 30
7.5 Initialization and Cleanup . . . 31
7.5.1 Detailed Description . . . 31
7.5.2 Function Documentation . . . 31
7.5.2.1 nvmlInit . . . 31
7.5.2.2 nvmlShutdown . . . 31
7.6 Error reporting . . . 32
7.6.1 Detailed Description . . . 32
7.6.2 Function Documentation . . . 32
7.6.2.1 nvmlErrorString . . . 32
7.7 Constants . . . 33
7.7.1 Define Documentation . . . 33
7.7.1.1 NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE . . . 33
7.7.1.2 NVML_DEVICE_NAME_BUFFER_SIZE . . . 33
7.7.1.3 NVML_DEVICE_SERIAL_BUFFER_SIZE . . . 33
7.7.1.4 NVML_DEVICE_UUID_BUFFER_SIZE . . . 33
7.7.1.5 NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE . . . 33
7.7.1.6 NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE . . . 33
7.7.1.7 NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE . . . 33

7.8 System Queries . . . 34
7.8.1 Detailed Description . . . 34
7.8.2 Function Documentation . . . 34
7.8.2.1 nvmlSystemGetDriverVersion . . . 34
7.8.2.2 nvmlSystemGetNVMLVersion . . . 34
7.8.2.3 nvmlSystemGetProcessName . . . 35
7.9 Unit Queries . . . 36
7.9.1 Detailed Description . . . 36
7.9.2 Function Documentation . . . 36
7.9.2.1 nvmlSystemGetHicVersion . . . 36
7.9.2.2 nvmlUnitGetCount . . . 36
7.9.2.3 nvmlUnitGetDevices . . . 37
7.9.2.4 nvmlUnitGetFanSpeedInfo . . . 37
7.9.2.5 nvmlUnitGetHandleByIndex . . . 38
7.9.2.6 nvmlUnitGetLedState . . . 38
7.9.2.7 nvmlUnitGetPsuInfo . . . 38
7.9.2.8 nvmlUnitGetTemperature . . . 39
7.9.2.9 nvmlUnitGetUnitInfo . . . 39
7.10 Device Queries . . . 40
7.10.1 Detailed Description . . . 41
7.10.2 Function Documentation . . . 41
7.10.2.1 nvmlDeviceGetApplicationsClock . . . 41
7.10.2.2 nvmlDeviceGetClockInfo . . . 42
7.10.2.3 nvmlDeviceGetComputeMode . . . 42
7.10.2.4 nvmlDeviceGetComputeRunningProcesses . . . 43
7.10.2.5 nvmlDeviceGetCount . . . 43
7.10.2.6 nvmlDeviceGetCurrentClocksThrottleReasons . . . 44
7.10.2.7 nvmlDeviceGetCurrPcieLinkGeneration . . . 44
7.10.2.8 nvmlDeviceGetCurrPcieLinkWidth . . . 45
7.10.2.9 nvmlDeviceGetDetailedEccErrors . . . 45
7.10.2.10 nvmlDeviceGetDisplayMode . . . 46
7.10.2.11 nvmlDeviceGetDriverModel . . . 46
7.10.2.12 nvmlDeviceGetEccMode . . . 47
7.10.2.13 nvmlDeviceGetFanSpeed . . . 47
7.10.2.14 nvmlDeviceGetGpuOperationMode . . . 48
7.10.2.15 nvmlDeviceGetHandleByIndex . . . 48
7.10.2.16 nvmlDeviceGetHandleByPciBusId . . . 49

7.10.2.17 nvmlDeviceGetHandleBySerial . . . 49
7.10.2.18 nvmlDeviceGetHandleByUUID . . . 50
7.10.2.19 nvmlDeviceGetInforomConfigurationChecksum . . . 50
7.10.2.20 nvmlDeviceGetInforomImageVersion . . . 51
7.10.2.21 nvmlDeviceGetInforomVersion . . . 51
7.10.2.22 nvmlDeviceGetMaxClockInfo . . . 52
7.10.2.23 nvmlDeviceGetMaxPcieLinkGeneration . . . 53
7.10.2.24 nvmlDeviceGetMaxPcieLinkWidth . . . 53
7.10.2.25 nvmlDeviceGetMemoryErrorCounter . . . 53
7.10.2.26 nvmlDeviceGetMemoryInfo . . . 54
7.10.2.27 nvmlDeviceGetName . . . 55
7.10.2.28 nvmlDeviceGetPciInfo . . . 55
7.10.2.29 nvmlDeviceGetPerformanceState . . . 55
7.10.2.30 nvmlDeviceGetPersistenceMode . . . 56
7.10.2.31 nvmlDeviceGetPowerManagementDefaultLimit . . . 56
7.10.2.32 nvmlDeviceGetPowerManagementLimit . . . 57
7.10.2.33 nvmlDeviceGetPowerManagementLimitConstraints . . . 57
7.10.2.34 nvmlDeviceGetPowerManagementMode . . . 58
7.10.2.35 nvmlDeviceGetPowerState . . . 59
7.10.2.36 nvmlDeviceGetPowerUsage . . . 59
7.10.2.37 nvmlDeviceGetSerial . . . 60
7.10.2.38 nvmlDeviceGetSupportedClocksThrottleReasons . . . 60
7.10.2.39 nvmlDeviceGetSupportedGraphicsClocks . . . 61
7.10.2.40 nvmlDeviceGetSupportedMemoryClocks . . . 61
7.10.2.41 nvmlDeviceGetTemperature . . . 62
7.10.2.42 nvmlDeviceGetTotalEccErrors . . . 62
7.10.2.43 nvmlDeviceGetUtilizationRates . . . 63
7.10.2.44 nvmlDeviceGetUUID . . . 63
7.10.2.45 nvmlDeviceGetVbiosVersion . . . 64
7.10.2.46 nvmlDeviceOnSameBoard . . . 64
7.10.2.47 nvmlDeviceResetApplicationsClocks . . . 65
7.10.2.48 nvmlDeviceValidateInforom . . . 65
7.11 Unit Commands . . . 66
7.11.1 Detailed Description . . . 66
7.11.2 Function Documentation . . . 66
7.11.2.1 nvmlUnitSetLedState . . . 66
7.12 Device Commands . . . 67

7.12.1 Detailed Description . . . 67
7.12.2 Function Documentation . . . 67
7.12.2.1 nvmlDeviceClearEccErrorCounts . . . 67
7.12.2.2 nvmlDeviceSetApplicationsClocks . . . 68
7.12.2.3 nvmlDeviceSetComputeMode . . . 68
7.12.2.4 nvmlDeviceSetDriverModel . . . 69
7.12.2.5 nvmlDeviceSetEccMode . . . 70
7.12.2.6 nvmlDeviceSetGpuOperationMode . . . 70
7.12.2.7 nvmlDeviceSetPersistenceMode . . . 71
7.12.2.8 nvmlDeviceSetPowerManagementLimit . . . 72
7.13 Event Handling Methods . . . 73
7.13.1 Detailed Description . . . 73
7.13.2 Typedef Documentation . . . 73
7.13.2.1 nvmlEventSet_t . . . 73
7.13.3 Function Documentation . . . 73
7.13.3.1 nvmlDeviceGetSupportedEventTypes . . . 73
7.13.3.2 nvmlDeviceRegisterEvents . . . 74
7.13.3.3 nvmlEventSetCreate . . . 75
7.13.3.4 nvmlEventSetFree . . . 75
7.13.3.5 nvmlEventSetWait . . . 75
7.14 NvmlClocksThrottleReasons . . . 77
7.14.1 Define Documentation . . . 77
7.14.1.1 nvmlClocksThrottleReasonAll . . . 77
7.14.1.2 nvmlClocksThrottleReasonGpuIdle . . . 77
7.14.1.3 nvmlClocksThrottleReasonHwSlowdown . . . 77
7.14.1.4 nvmlClocksThrottleReasonNone . . . 78
7.14.1.5 nvmlClocksThrottleReasonSwPowerCap . . . 78
7.14.1.6 nvmlClocksThrottleReasonUnknown . . . 78
7.14.1.7 nvmlClocksThrottleReasonUserDefinedClocks . . . 78

8 Data Structure Documentation 79

8.1 nvmlEccErrorCounts_t Struct Reference . . . 79
8.1.1 Detailed Description . . . 79
8.2 nvmlEventData_t Struct Reference . . . 80
8.2.1 Detailed Description . . . 80
8.3 nvmlHwbcEntry_t Struct Reference . . . 81
8.3.1 Detailed Description . . . 81

8.4 nvmlLedState_t Struct Reference 82
8.4.1 Detailed Description 82
8.5 nvmlMemory_t Struct Reference 83
8.5.1 Detailed Description 83
8.6 nvmlPciInfo_t Struct Reference 84
8.6.1 Detailed Description 84
8.7 nvmlProcessInfo_t Struct Reference 85
8.7.1 Detailed Description 85
8.8 nvmlPSUInfo_t Struct Reference 86
8.8.1 Detailed Description 86
8.9 nvmlUnitFanInfo_t Struct Reference 87
8.9.1 Detailed Description 87
8.10 nvmlUnitFanSpeeds_t Struct Reference 88
8.10.1 Detailed Description 88
8.11 nvmlUnitInfo_t Struct Reference 89
8.11.1 Detailed Description 89
8.12 nvmlUtilization_t Struct Reference 90
8.12.1 Detailed Description 90

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: S2050, C2050, C2070, C2075, M2050, M2070, M2075, M2090, X2070, X2090, K10, K20, K20X
  - NVIDIA Quadro ®Line: 4000, 5000, 6000, 7000, M2070-Q, 600, 2000, 3000M and 410
  - NVIDIA GeForce ®Line: None
- Limited Support
  - NVIDIA Tesla ™Line: S1070, C1060, M1060
  - 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			C2050	C2070	C2075	M2050	M2070	M2075	M2090	S2050	X2070	X2090	K10	K20
Board Serial Number			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GPU UUID			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
VBios Version			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Inforom	Image Version		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
	OEM Object Version		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	ECC Object Version		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Power Object Version		✗	✗	✓	✗	✗	✓	✓	✗	✗	✓	✗	✗
	Checksum		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PCI Info			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Compute Mode			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Display Mode			✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Persistence Mode (Linux-Only)			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ECC Mode			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ECC Error Counter Type	Device Memory		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Register File		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
	L1 Cache		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
	L2 Cache		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓
	Texture Memory		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Driver Model (Windows-Only)			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Fan Speed			✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
GPU Temperature			✓	✓	✓	✗	✗	✗	✗	✓	✗	✗	✓	✓
Memory Usage			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Power Readings	Power Management Enabled		✗	✗	✓	✗	✗	✓	✓	✗	✗	✓	✓	✓
	Current Power Draw		✗	✗	✓	✗	✗	✓	✓	✗	✗	✓	✓	✓
	Power Draw Limit		✗	✗	✓	✗	✗	✓	✓	✗	✗	✓	✓	✓
	Default Power Limit		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
	Power Limit Constraints		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Clock Speeds	Current	Graphics	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		SM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Memory	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Application Clocks	Graphics	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
		Memory	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
	Max	Graphics	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		SM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Memory	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Listing Supported Clock combinations		✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Utilization Rates			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ECC Errors	Volatile	Location-Based	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Total	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Aggregate	Location-Based	✗	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓
		Total	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Performance State			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Process Info			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GPU Operation Mode (GOM)			✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Figure 1.1: This chart shows which features are reported for each Fermi and Kepler architecture GPU product.

Queries			4000	5000	6000	Quadro Plex 7000	M2070-Q	600	2000	3000M	410
Board Serial Number			✓	✓	✓	✓	✓	✗	✗	✗	✗
GPU UUID			✓	✓	✓	✓	✓	✓	✓	✓	✓
VBios Version			✓	✓	✓	✓	✓	✓	✓	✓	✓
Inforom	Image Version		✗	✗	✗	✗	✗	✗	✗	✗	✗
	OEM Object Version		✓	✓	✓	✓	✓	✗	✗	✗	✗
	ECC Object Version		✗	✓	✓	✓	✓	✗	✗	✗	✗
	Power Object Version		✗	✗	✗	✓	✗	✗	✗	✗	✗
	Checksum		✓	✓	✓	✓	✓	✓	✓	✓	✓
PCI Info			✓	✓	✓	✓	✓	✓	✓	✓	✓
Compute Mode			✓	✓	✓	✓	✓	✓	✓	✓	✓
Display Mode			✓	✓	✓	✓	✗	✓	✓	✓	✓
Persistence Mode (Linux-Only)			✓	✓	✓	✓	✓	✓	✓	✓	✓
ECC Mode			✗	✓	✓	✓	✓	✗	✗	✗	✗
ECC Error Counter Type	Device Memory		✗	✓	✓	✓	✓	✗	✗	✗	✗
	Register File		✗	✓	✓	✓	✓	✗	✗	✗	✗
	L1 Cache		✗	✓	✓	✓	✓	✗	✗	✗	✗
	L2 Cache		✗	✓	✓	✓	✓	✗	✗	✗	✗
	Texture Memory		✗	✗	✗	✗	✗	✗	✗	✗	✗
Driver Model (Windows-Only)			✓	✓	✓	✓	✓	✓	✓	✓	✓
Fan Speed			✓	✓	✓	✓	✗	✓	✓	✓	✓
GPU Temperature			✓	✓	✓	✓	✗	✓	✓	✓	✓
Memory Usage			✓	✓	✓	✓	✓	✓	✓	✓	✓
Power Readings	Power Management Enabled		✗	✗	✗	✓	✗	✗	✗	✗	✗
	Current Power Draw		✗	✗	✗	✓	✗	✗	✗	✗	✗
	Power Draw Limit		✗	✗	✗	✓	✗	✗	✗	✗	✗
	Default Power Limit		✗	✗	✗	✗	✗	✗	✗	✗	✗
	Power Limit Constraints		✗	✗	✗	✗	✗	✗	✗	✗	✗
Clock Speeds	Current	Graphics	✓	✓	✓	✓	✓	✓	✓	✓	✓
		SM	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Memory	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Application Clocks	Graphics	✗	✗	✗	✗	✗	✗	✗	✗	✗
		Memory	✗	✗	✗	✗	✗	✗	✗	✗	✗
	Max	Graphics	✓	✓	✓	✓	✓	✓	✓	✓	✓
		SM	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Memory	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Listing Supported Clock combinations		✗	✗	✗	✗	✗	✗	✗	✗	✗
Utilization Rates			✓	✓	✓	✓	✓	✓	✓	✓	✓
ECC Errors	Volatile	Location-Based	✗	✓	✓	✓	✓	✗	✗	✗	✗
		Total	✗	✓	✓	✓	✓	✗	✗	✗	✗
	Aggregate	Location-Based	✗	✗	✗	✓	✓	✗	✗	✗	✗
		Total	✗	✓	✓	✓	✓	✗	✗	✗	✗
Performance State			✓	✓	✓	✓	✓	✓	✓	✓	✓
Process Info			✓	✓	✓	✓	✓	✓	✓	✓	✓
GPU Operation Mode (GOM)			✗	✗	✗	✗	✗	✗	✗	✗	✗

Figure 1.2: This chart shows which features are reported for each Quadro and T10 GPU product.

Commands	C2050	C2070	C2075	M2050	M2070	M2075	M2090	S2050	X2070	X2090	K10	K20
Set Compute Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set Persistence Mode (Linux-Only)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set Display Model (Win7-Only)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set ECC Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Clear ECC Errors	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Reset GPU (Linux-Only)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set Application Clocks	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Set Power Management Limit	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Set GPU Operation Mode (GOM)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Figure 1.3: This chart shows which commands are available for each Fermi and Kepler architecture GPU product.

Commands	4000	5000	6000	Quadro Plex 7000	M2070-Q	600	2000	3000M	410
Set Compute Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set Persistence Mode (Linux-Only)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set Display Model (Win7-Only)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set ECC Mode	✗	✓	✓	✓	✓	✗	✗	✗	✗
Clear ECC Errors	✗	✓	✓	✓	✓	✗	✗	✗	✗
Reset GPU (Linux-Only)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Set Application Clocks	✗	✗	✗	✗	✗	✗	✗	✗	✗
Set Power Management Limit	✗	✗	✗	✗	✗	✗	✗	✗	✗
Set GPU Operation Mode (GOM)	✗	✗	✗	✗	✗	✗	✗	✗	✗

Figure 1.4: This chart shows which commands are available for each Quadro and T10 GPU product.

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 1.5: 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.

Chapter 2

Known issues in the current version of NVML library

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.

Chapter 3

Change log of NVML library

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

3.1 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

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

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

- Added nvmlDeviceGetInforomConfigurationChecksum and nvmlDeviceValidateInforom
- Added new error return value for initialization failure due to kernel module not receiving interrupts
- Added nvmlDeviceSetApplicationsClocks, nvmlDeviceGetApplicationsClock, nvmlDeviceResetApplicationsClocks
- Added nvmlDeviceGetSupportedMemoryClocks and nvmlDeviceGetSupportedGraphicsClocks
- Added nvmlDeviceGetPowerManagementLimitConstraints, nvmlDeviceGetPowerManagementDefaultLimit and nvmlDeviceSetPowerManagementLimit
- Added nvmlDeviceGetInforomImageVersion
- Expanded nvmlDeviceGetUUID to support all CUDA capable GPUs
- Deprecated nvmlDeviceGetDetailedEccErrors in favor of nvmlDeviceGetMemoryErrorCounter
- Added NVML_MEMORY_LOCATION_TEXTURE_MEMORY to support reporting of texture memory error counters
- Added nvmlDeviceGetCurrentClocksThrottleReasons and nvmlDeviceGetSupportedClocksThrottleReasons
- NVML_CLOCK_SM is now also reported on supported Kepler devices.
- Dropped support for GT200 based Tesla brand GPUs: C1060, M1060 and S1070

3.3 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 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

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

Chapter 4

Deprecated List

Class nvmlEccErrorCounts_t Different GPU families can have different memory error counters See nvmlDeviceGetMemoryErrorCounter

Global NVML_DOUBLE_BIT_ECC Mapped to NVML_MEMORY_ERROR_TYPE_UNCORRECTED

Global NVML_SINGLE_BIT_ECC Mapped to NVML_MEMORY_ERROR_TYPE_CORRECTED

Global nvmlEccBitType_t See nvmlMemoryErrorType_t for a more flexible type

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

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 5

Module Index

5.1 Modules

Here is a list of all modules:

Device Structs 21
Device Enums 22
Unit Structs 29
Initialization and Cleanup 31
Error reporting 32
Constants 33
System Queries 34
Unit Queries 36
Device Queries 40
Unit Commands 66
Device Commands 67
Event Handling Methods 73
    Event Types 30
NvmlClocksThrottleReasons 77

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

nvmlEccErrorCounts_t 79
nvmlEventData_t 80
nvmlHwbcEntry_t 81
nvmlLedState_t 82
nvmlMemory_t 83
nvmlPciInfo_t 84
nvmlProcessInfo_t 85
nvmlPSUInfo_t 86
nvmlUnitFanInfo_t 87
nvmlUnitFanSpeeds_t 88
nvmlUnitInfo_t 89
nvmlUtilization_t 90

Chapter 7

Module Documentation

7.1 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)

7.1.1 Define Documentation

7.1.1.1 #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.

7.2 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.

- #define nvmlEccBitType_t nvmlMemoryErrorType_t
- #define NVML_SINGLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_CORRECTED
- #define NVML_DOUBLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_UNCORRECTED

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 nvmlMemoryErrorType_t {
  NVML_MEMORY_ERROR_TYPE_CORRECTED = 0,
  NVML_MEMORY_ERROR_TYPE_UNCORRECTED = 1,
  NVML_MEMORY_ERROR_TYPE_COUNT }
- 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 nvmlGpuOperationMode_t {
  NVML_GOM_ALL_ON = 0,
  NVML_GOM_COMPUTE = 1,
  NVML_GOM_LOW_DP = 2 }
- enum nvmlInforomObject_t {
  NVML_INFOROM_OEM = 0,
  NVML_INFOROM_ECC = 1,
  NVML_INFOROM_POWER = 2,
  NVML_INFOROM_COUNT }
- 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_IRQ_ISSUE = 11,
  NVML_ERROR_LIBRARY_NOT_FOUND = 12,
  NVML_ERROR_FUNCTION_NOT_FOUND = 13,
  NVML_ERROR_CORRUPTED_INFOROM = 14,
  NVML_ERROR_UNKNOWN = 999 }

- enum nvmlMemoryLocation_t {
  NVML_MEMORY_LOCATION_L1_CACHE = 0,
  NVML_MEMORY_LOCATION_L2_CACHE = 1,
  NVML_MEMORY_LOCATION_DEVICE_MEMORY = 2,
  NVML_MEMORY_LOCATION_REGISTER_FILE = 3,
  NVML_MEMORY_LOCATION_TEXTURE_MEMORY = 4,
  NVML_MEMORY_LOCATION_COUNT }

7.2.1 Define Documentation

7.2.1.1 #define NVML_DOUBLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_UNCORRECTED

Double bit ECC errors

Deprecated

Mapped to NVML_MEMORY_ERROR_TYPE_UNCORRECTED

7.2.1.2 #define NVML_SINGLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_CORRECTED

Single bit ECC errors

Deprecated

Mapped to NVML_MEMORY_ERROR_TYPE_CORRECTED

7.2.1.3 #define nvmlEccBitType_t nvmlMemoryErrorType_t

ECC bit types.

Deprecated

See nvmlMemoryErrorType_t for a more flexible type

7.2.2 Enumeration Type Documentation

7.2.2.1 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.

7.2.2.2 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.

7.2.2.3 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.

7.2.2.4 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).

7.2.2.5 enum nvmlEnableState_t

Generic enable/disable enum.

Enumerator:

NVML_FEATURE_DISABLED Feature disabled.

NVML_FEATURE_ENABLED Feature enabled.

7.2.2.6 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.

Enumerator:

NVML_GOM_ALL_ON Everything is enabled and running at full speed.

NVML_GOM_COMPUTE Designed for running only compute tasks. Graphics operations < are not allowed.

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

7.2.2.7 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.

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

7.2.2.8 enum nvmlMemoryErrorType_t

Memory error types

Enumerator:

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

NVML_MEMORY_ERROR_TYPE_UNCORRECTED A memory error that was not corrected
For 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.

7.2.2.9 enum nvmlMemoryLocation_t

Memory locations

See nvmlDeviceGetMemoryErrorCounter

Enumerator:

NVML_MEMORY_LOCATION_L1_CACHE GPU L1 Cache.

NVML_MEMORY_LOCATION_L2_CACHE GPU L2 Cache.

NVML_MEMORY_LOCATION_DEVICE_MEMORY GPU Device Memory.

NVML_MEMORY_LOCATION_REGISTER_FILE GPU Register File.

NVML_MEMORY_LOCATION_TEXTURE_MEMORY GPU Texture Memory.

NVML_MEMORY_LOCATION_COUNT This counts the number of memory locations the driver knows about.

7.2.2.10 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.

7.2.2.11 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_IRQ_ISSUE NVIDIA Kernel detected an interrupt issue with a GPU.

NVML_ERROR_LIBRARY_NOT_FOUND NVML Shared Library couldn't be found or loaded.

NVML_ERROR_FUNCTION_NOT_FOUND Local version of NVML doesn't implement this function.

NVML_ERROR_CORRUPTED_INFOROM infoROM is corrupted

NVML_ERROR_UNKNOWN An internal driver error occurred.

7.2.2.12 enum nvmlTemperatureSensors_t

Temperature sensors.

Enumerator:

NVML_TEMPERATURE_GPU Temperature sensor for the GPU die.

7.3 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 }

7.3.1 Enumeration Type Documentation

7.3.1.1 enum nvmlFanState_t

Fan state enum.

Enumerator:

NVML_FAN_NORMAL Fan is working properly.

NVML_FAN_FAILED Fan has failed.

7.3.1.2 enum nvmlLedColor_t

Led color enum.

Enumerator:

NVML_LED_COLOR_GREEN GREEN, indicates good health.

NVML_LED_COLOR_AMBER AMBER, indicates problem.

7.4 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 nvmlEventTypeClock 0x0000000000000010LL

  Event about clock changes.

- #define nvmlEventTypeNone 0x0000000000000000LL

  Mask with no events.

- #define nvmlEventTypeAll

  Mask of all events.

7.4.1 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

7.4.2 Define Documentation

7.4.2.1 #define nvmlEventTypeClock 0x0000000000000010LL

Kepler only

7.4.2.2 #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.

7.5 Initialization and Cleanup

Functions

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

7.5.1 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.

7.5.2 Function Documentation

7.5.2.1 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

7.5.2.2 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

7.6 Error reporting

Functions

- const DECLDIR char ∗ nvmlErrorString (nvmlReturn_t result)

7.6.1 Detailed Description

This chapter describes helper functions for error reporting routines.

7.6.2 Function Documentation

7.6.2.1 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.

7.7 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

7.7.1 Define Documentation

7.7.1.1 #define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16

Buffer size guaranteed to be large enough for nvmlDeviceGetInforomVersion and nvmlDeviceGetInforomImageVersion

7.7.1.2 #define NVML_DEVICE_NAME_BUFFER_SIZE 64

Buffer size guaranteed to be large enough for nvmlDeviceGetName

7.7.1.3 #define NVML_DEVICE_SERIAL_BUFFER_SIZE 30

Buffer size guaranteed to be large enough for nvmlDeviceGetSerial

7.7.1.4 #define NVML_DEVICE_UUID_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlDeviceGetUUID

7.7.1.5 #define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32

Buffer size guaranteed to be large enough for nvmlDeviceGetVbiosVersion

7.7.1.6 #define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlSystemGetDriverVersion

7.7.1.7 #define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlSystemGetNVMLVersion

7.8 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)

7.8.1 Detailed Description

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

7.8.2 Function Documentation

7.8.2.1 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

7.8.2.2 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

7.8.2.3 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

7.9 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)

7.9.1 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().

7.9.2 Function Documentation

7.9.2.1 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

7.9.2.2 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

7.9.2.3 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

7.9.2.4 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

7.9.2.5 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

7.9.2.6 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()

7.9.2.7 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

7.9.2.8 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

7.9.2.9 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

7.10 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 nvmlDeviceGetInforomImageVersion (nvmlDevice_t device, char *version, unsigned int length)
- nvmlReturn_t DECLDIR nvmlDeviceGetInforomConfigurationChecksum (nvmlDevice_t device, unsigned int *checksum)
- nvmlReturn_t DECLDIR nvmlDeviceValidateInforom (nvmlDevice_t device)
- 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 nvmlDeviceGetApplicationsClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int *clockMHz)
- nvmlReturn_t DECLDIR nvmlDeviceResetApplicationsClocks (nvmlDevice_t device)
- nvmlReturn_t DECLDIR nvmlDeviceGetSupportedMemoryClocks (nvmlDevice_t device, unsigned int *count, unsigned int *clocksMHz)
- nvmlReturn_t DECLDIR nvmlDeviceGetSupportedGraphicsClocks (nvmlDevice_t device, unsigned int memoryClockMHz, unsigned int *count, unsigned int *clocksMHz)
- 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 nvmlDeviceGetCurrentClocksThrottleReasons (nvmlDevice_t device, unsigned long long *clocksThrottleReasons)
- nvmlReturn_t DECLDIR nvmlDeviceGetSupportedClocksThrottleReasons (nvmlDevice_t device, unsigned long long *supportedClocksThrottleReasons)
- 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 nvmlDeviceGetPowerManagementLimitConstraints (nvmlDevice_t device, unsigned int *minLimit, unsigned int *maxLimit)
- nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementDefaultLimit (nvmlDevice_t device, unsigned int *defaultLimit)
- nvmlReturn_t DECLDIR nvmlDeviceGetPowerUsage (nvmlDevice_t device, unsigned int *power)
- nvmlReturn_t DECLDIR nvmlDeviceGetGpuOperationMode (nvmlDevice_t device, nvmlGpuOperationMode_t *current, nvmlGpuOperationMode_t *pending)
- 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, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, unsigned long long *eccCounts)
- nvmlReturn_t DECLDIR nvmlDeviceGetDetailedEccErrors (nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, nvmlEccErrorCounts_t *eccCounts)
- nvmlReturn_t DECLDIR nvmlDeviceGetMemoryErrorCounter (nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, nvmlMemoryLocation_t locationType, unsigned long long *count)
- 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)

7.10.1 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().

7.10.2 Function Documentation

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

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

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

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 new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_FOUND if the max clock limit is not set
- NVML_ERROR_INVALID_ARGUMENT if *device* is invalid or *clockMHz* is NULL or *clockType* is invalid
- NVML_ERROR_NOT_SUPPORTED if the device cannot report the specified clock
- NVML_ERROR_UNKNOWN on any unexpected error

7.10.2.2 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

7.10.2.3 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()

7.10.2.4 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

7.10.2.5 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

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

Retrieves current clocks throttling reasons.

For Tesla ™products from Kepler family.

Note:

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

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 returned successfully
- 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_UNKNOWN on any unexpected error

See also:

NvmlClocksThrottleReasons
nvmlDeviceGetSupportedClocksThrottleReasons

7.10.2.7 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

7.10.2.8 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

7.10.2.9 nvmlReturn_t DECLDIR nvmlDeviceGetDetailedEccErrors (nvmlDevice_t *device*, nvmlMemoryErrorType_t *errorType*, nvmlEccCounterType_t *counterType*, nvmlEccErrorCounts_t * *eccCounts*)

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 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 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.

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_UNKNOWN on any unexpected error

See also:

nvmlDeviceClearEccErrorCounts()

7.10.2.10 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

7.10.2.11 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()

7.10.2.12 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()

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

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%.

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

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

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. Not supported on Quadro ®and Tesla ™C-class products.

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_UNKNOWN on any unexpected error

See also:

nvmlGpuOperationMode_t
nvmlDeviceSetGpuOperationMode

7.10.2.15 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

7.10.2.16 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

7.10.2.17 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

7.10.2.18 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

7.10.2.19 nvmlReturn_t DECLDIR nvmlDeviceGetInforomConfigurationChecksum (nvmlDevice_t *device*, unsigned int * *checksum*)

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

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

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)

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_UNKNOWN on any unexpected error

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

Retrieves the global infoROM image version

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

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.

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_UNKNOWN on any unexpected error

See also:

nvmlDeviceGetInforomVersion

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

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

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

See also:

nvmlDeviceGetInforomImageVersion

7.10.2.22 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

7.10.2.23 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

7.10.2.24 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

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

Retrieves the requested memory error counter for the device.

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

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

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.

Parameters:

device The identifier of the target device

errorType Flag that specifies the type of error.

counterType Flag that specifies the counter-type of the errors.

locationType Specifies the location of the counter.

count Reference in which to return the ECC counter

Returns:

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

7.10.2.26 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

7.10.2.27 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

7.10.2.28 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

7.10.2.29 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

7.10.2.30 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()

7.10.2.31 nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementDefaultLimit (nvmlDevice_t *device*, unsigned int * *defaultLimit*)

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 Tesla ™and Quadro ®products from the Kepler family.

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_UNKNOWN on any unexpected error

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

Retrieves the power management limit 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.

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 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_UNKNOWN on any unexpected error

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

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

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

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* has 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_UNKNOWN on any unexpected error

See also:

nvmlDeviceSetPowerManagementLimit

7.10.2.34 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

7.10.2.35 nvmlReturn_t DECLDIR nvmlDeviceGetPowerState (nvmlDevice_t *device*, nvmlPstates_t * *pState*)

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

Retrieve 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

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

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

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.

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.

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

7.10.2.37 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

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

Retrieves bitmask of supported clocks throttle reasons that can be returned by nvmlDeviceGetCurrentClocksThrot-tleReasons

For all devices

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 returned successfully
- 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_UNKNOWN on any unexpected error

See also:

NvmlClocksThrottleReasons
nvmlDeviceGetCurrentClocksThrottleReasons

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

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

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

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 new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_FOUND if the max clock limit is not set
- NVML_ERROR_INVALID_ARGUMENT if *device* is invalid or *clock* is NULL or *clockType* is invalid
- NVML_ERROR_NOT_SUPPORTED if the device cannot report the specified clock
- NVML_ERROR_INSUFFICIENT_SIZE if *count* is too small
- NVML_ERROR_UNKNOWN on any unexpected error

See also:

nvmlDeviceSetApplicationsClocks
nvmlDeviceGetSupportedMemoryClocks

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

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

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

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 new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_FOUND if the max clock limit is not set
- NVML_ERROR_INVALID_ARGUMENT if *device* is invalid or *clock* is NULL or *clockType* is invalid
- NVML_ERROR_NOT_SUPPORTED if the device cannot report the specified clock
- NVML_ERROR_INSUFFICIENT_SIZE if *count* is too small
- NVML_ERROR_UNKNOWN on any unexpected error

See also:

nvmlDeviceSetApplicationsClocks
nvmlDeviceGetSupportedGraphicsClocks

7.10.2.41 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

7.10.2.42 nvmlReturn_t DECLDIR nvmlDeviceGetTotalEccErrors (nvmlDevice_t *device*, nvmlMemoryErrorType_t *errorType*, 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 nvmlMemoryErrorType_t for a description of available error types.

See nvmlEccCounterType_t for a description of available counter types.

Parameters:

device The identifier of the target device

errorType Flag that specifies the type of the errors.

counterType Flag that specifies the counter-type of the errors.

eccCounts Reference in which to return the specified ECC errors

Returns:

- NVML_SUCCESS if *eccCounts* has been set

- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if *device*, *errorType* or *counterType* is invalid, or *eccCounts* is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

See also:

nvmlDeviceClearEccErrorCounts()

7.10.2.43 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

7.10.2.44 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 all CUDA capable GPUs.

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

7.10.2.45 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

7.10.2.46 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

7.10.2.47 nvmlReturn_t DECLDIR nvmlDeviceResetApplicationsClocks (nvmlDevice_t *device*)

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

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_NOT_FOUND if the max clock limit is not set
- NVML_ERROR_INVALID_ARGUMENT if *device* is invalid
- NVML_ERROR_NOT_SUPPORTED if the device cannot perform this action
- NVML_ERROR_UNKNOWN on any unexpected error

7.10.2.48 nvmlReturn_t DECLDIR nvmlDeviceValidateInforom (nvmlDevice_t *device*)

Reads the infoROM from the flash and verifies the checksums.

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

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_UNKNOWN on any unexpected error

7.11 Unit Commands

Functions

- nvmlReturn_t DECLDIR nvmlUnitSetLedState (nvmlUnit_t unit, nvmlLedColor_t color)

7.11.1 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.

7.11.2 Function Documentation

7.11.2.1 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()

7.12 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)
- nvmlReturn_t DECLDIR nvmlDeviceSetApplicationsClocks (nvmlDevice_t device, unsigned int memClockMHz, unsigned int graphicsClockMHz)
- nvmlReturn_t DECLDIR nvmlDeviceSetPowerManagementLimit (nvmlDevice_t device, unsigned int limit)
- nvmlReturn_t DECLDIR nvmlDeviceSetGpuOperationMode (nvmlDevice_t device, nvmlGpuOperationMode_t mode)

7.12.1 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.

7.12.2 Function Documentation

7.12.2.1 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()

7.12.2.2 nvmlReturn_t DECLDIR nvmlDeviceSetApplicationsClocks (nvmlDevice_t *device*, unsigned int *memClockMHz*, unsigned int *graphicsClockMHz*)

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

Parameters:

device The identifier of the target device

memClockMHz Requested memory clock in MHz

graphicsClockMHz Requested graphics clock in MHz

Returns:

- NVML_SUCCESS if new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if *device* is invalid or *memClockMHz* and *graphicsClockMHz* is not a valid clock combination
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

7.12.2.3 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()

7.12.2.4 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.

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

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()

7.12.2.5 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()

7.12.2.6 nvmlReturn_t DECLDIR nvmlDeviceSetGpuOperationMode (nvmlDevice_t *device*, nvmlGpuOperationMode_t *mode*)

Sets new GOM. See *nvmlGpuOperationMode_t* for details.

For GK110 M-class and X-class Tesla ™products from the Kepler family. 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.

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_UNKNOWN on any unexpected error

See also:

nvmlGpuOperationMode_t
nvmlDeviceGetGpuOperationMode

7.12.2.7 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()

7.12.2.8 nvmlReturn_t DECLDIR nvmlDeviceSetPowerManagementLimit (nvmlDevice_t *device*, unsigned int *limit*)

Set new power limit of this device.

For Tesla ™and Quadro ®products from the Kepler family. Requires root/admin permissions.

See nvmlDeviceGetPowerManagementLimitConstraints to check the allowed ranges of values.

Note:

Limit is not persistent across reboots or driver unloads. Enable persistent mode to prevent driver from unloading when no application is using the device.

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_UNKNOWN on any unexpected error

See also:

nvmlDeviceGetPowerManagementLimitConstraints
nvmlDeviceGetPowerManagementDefaultLimit

7.13 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)

7.13.1 Detailed Description

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

7.13.2 Typedef Documentation

7.13.2.1 typedef struct nvmlEventSet_st* nvmlEventSet_t

Handle to an event set

7.13.3 Function Documentation

7.13.3.1 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

7.13.3.2 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

7.13.3.3 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

7.13.3.4 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

7.13.3.5 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 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_UNKNOWN on any unexpected error

See also:

Event Types
nvmlDeviceRegisterEvents

7.14 NvmlClocksThrottleReasons

Defines

- #define nvmlClocksThrottleReasonGpuIdle 0x0000000000000001LL
- #define nvmlClocksThrottleReasonUserDefinedClocks 0x0000000000000002LL
- #define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL
- #define nvmlClocksThrottleReasonHwSlowdown 0x0000000000000008LL
- #define nvmlClocksThrottleReasonUnknown 0x8000000000000000LL
- #define nvmlClocksThrottleReasonNone 0x0000000000000000LL
- #define nvmlClocksThrottleReasonAll

7.14.1 Define Documentation

7.14.1.1 #define nvmlClocksThrottleReasonAll

Value:

```
(nvmlClocksThrottleReasonNone \
      | nvmlClocksThrottleReasonGpuIdle                       \
      | nvmlClocksThrottleReasonUserDefinedClocks             \
      | nvmlClocksThrottleReasonSwPowerCap                    \
      | nvmlClocksThrottleReasonHwSlowdown                    \
      | nvmlClocksThrottleReasonUnknown                       \
        )
```

Bit mask representing all supported clocks throttling reasons New reasons might be added to this list in the future

7.14.1.2 #define nvmlClocksThrottleReasonGpuIdle 0x0000000000000001LL

Nothing is running on the GPU and the clocks are dropping to Idle state

Note:

This limiter may be removed in a later release

7.14.1.3 #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
nvmlDeviceGetPowerUsage

7.14.1.4 #define nvmlClocksThrottleReasonNone 0x0000000000000000LL

Bit mask representing no clocks throttling

Clocks are as high as possible.

7.14.1.5 #define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL

SW Power Scaling algorithm is reducing the clocks below requested clocks

See also:

nvmlDeviceGetPowerUsage
nvmlDeviceSetPowerManagementLimit
nvmlDeviceGetPowerManagementLimit

7.14.1.6 #define nvmlClocksThrottleReasonUnknown 0x8000000000000000LL

Some other unspecified factor is reducing the clocks

7.14.1.7 #define nvmlClocksThrottleReasonUserDefinedClocks 0x0000000000000002LL

GPU clocks are limited by user specified limit

See also:

nvmlDeviceSetApplicationsClocks
nvmlDeviceGetApplicationsClock

Chapter 8

Data Structure Documentation

8.1 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.

8.1.1 Detailed Description

Detailed ECC error counts for a device.

Deprecated

Different GPU families can have different memory error counters See nvmlDeviceGetMemoryErrorCounter

8.2 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.

8.2.1 Detailed Description

Information about occurred event

8.3 nvmlHwbcEntry_t Struct Reference

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

8.3.1 Detailed Description

Description of HWBC entry

8.4 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.

8.4.1 Detailed Description

LED states for an S-class unit.

8.5 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.

8.5.1 Detailed Description

Memory allocation information for a device.

8.6 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.

8.6.1 Detailed Description

PCI information about a GPU device.

8.7 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.

8.7.1 Detailed Description

Information about running compute processes on the GPU

8.8 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).

8.8.1 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

8.9 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.

8.9.1 Detailed Description

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

8.10 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.

8.10.1 Detailed Description

Fan speed readings for an entire S-class unit.

8.11 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.

8.11.1 Detailed Description

Static S-class unit info.

8.12 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.

8.12.1 Detailed Description

Utilization information for a device.

Index

Constants, 33

Device Commands, 67
Device Enums, 22
Device Queries, 40
Device Structs, 21

Error reporting, 32
Event Handling Methods, 73
Event Types, 30

Initialization and Cleanup, 31

NVML_AGGREGATE_ECC
    nvmlDeviceEnumvs, 25
NVML_CLOCK_GRAPHICS
    nvmlDeviceEnumvs, 24
NVML_CLOCK_MEM
    nvmlDeviceEnumvs, 24
NVML_CLOCK_SM
    nvmlDeviceEnumvs, 24
NVML_COMPUTEMODE_DEFAULT
    nvmlDeviceEnumvs, 25
NVML_COMPUTEMODE_EXCLUSIVE_PROCESS
    nvmlDeviceEnumvs, 25
NVML_COMPUTEMODE_EXCLUSIVE_THREAD
    nvmlDeviceEnumvs, 25
NVML_COMPUTEMODE_PROHIBITED
    nvmlDeviceEnumvs, 25
NVML_DRIVER_WDDM
    nvmlDeviceEnumvs, 25
NVML_DRIVER_WDM
    nvmlDeviceEnumvs, 25
NVML_ERROR_ALREADY_INITIALIZED
    nvmlDeviceEnumvs, 27
NVML_ERROR_CORRUPTED_INFOROM
    nvmlDeviceEnumvs, 27
NVML_ERROR_DRIVER_NOT_LOADED
    nvmlDeviceEnumvs, 27
NVML_ERROR_FUNCTION_NOT_FOUND
    nvmlDeviceEnumvs, 27
NVML_ERROR_INSUFFICIENT_POWER
    nvmlDeviceEnumvs, 27
NVML_ERROR_INSUFFICIENT_SIZE
    nvmlDeviceEnumvs, 27
NVML_ERROR_INVALID_ARGUMENT
    nvmlDeviceEnumvs, 27
NVML_ERROR_IRQ_ISSUE
    nvmlDeviceEnumvs, 27
NVML_ERROR_LIBRARY_NOT_FOUND
    nvmlDeviceEnumvs, 27
NVML_ERROR_NO_PERMISSION
    nvmlDeviceEnumvs, 27
NVML_ERROR_NOT_FOUND
    nvmlDeviceEnumvs, 27
NVML_ERROR_NOT_SUPPORTED
    nvmlDeviceEnumvs, 27
NVML_ERROR_TIMEOUT
    nvmlDeviceEnumvs, 27
NVML_ERROR_UNINITIALIZED
    nvmlDeviceEnumvs, 27
NVML_ERROR_UNKNOWN
    nvmlDeviceEnumvs, 27
NVML_FAN_FAILED
    nvmlUnitStructs, 29
NVML_FAN_NORMAL
    nvmlUnitStructs, 29
NVML_FEATURE_DISABLED
    nvmlDeviceEnumvs, 25
NVML_FEATURE_ENABLED
    nvmlDeviceEnumvs, 25
NVML_GOM_ALL_ON
    nvmlDeviceEnumvs, 26
NVML_GOM_COMPUTE
    nvmlDeviceEnumvs, 26
NVML_GOM_LOW_DP
    nvmlDeviceEnumvs, 26
NVML_INFOROM_COUNT
    nvmlDeviceEnumvs, 26
NVML_INFOROM_ECC
    nvmlDeviceEnumvs, 26
NVML_INFOROM_OEM
    nvmlDeviceEnumvs, 26
NVML_INFOROM_POWER
    nvmlDeviceEnumvs, 26
NVML_LED_COLOR_AMBER
    nvmlUnitStructs, 29
NVML_LED_COLOR_GREEN
    nvmlUnitStructs, 29
NVML_MEMORY_ERROR_TYPE_CORRECTED
    nvmlDeviceEnumvs, 26

NVML_MEMORY_ERROR_TYPE_COUNT
nvmlDeviceEnumvs, 26
NVML_MEMORY_ERROR_TYPE_UNCORRECTED
nvmlDeviceEnumvs, 26
NVML_MEMORY_LOCATION_COUNT
nvmlDeviceEnumvs, 26
NVML_MEMORY_LOCATION_DEVICE_MEMORY
nvmlDeviceEnumvs, 26
NVML_MEMORY_LOCATION_L1_CACHE
nvmlDeviceEnumvs, 26
NVML_MEMORY_LOCATION_L2_CACHE
nvmlDeviceEnumvs, 26
NVML_MEMORY_LOCATION_REGISTER_FILE
nvmlDeviceEnumvs, 26
NVML_MEMORY_LOCATION_TEXTURE_-
MEMORY
nvmlDeviceEnumvs, 26
NVML_PSTATE_0
nvmlDeviceEnumvs, 27
NVML_PSTATE_1
nvmlDeviceEnumvs, 27
NVML_PSTATE_10
nvmlDeviceEnumvs, 27
NVML_PSTATE_11
nvmlDeviceEnumvs, 27
NVML_PSTATE_12
nvmlDeviceEnumvs, 27
NVML_PSTATE_13
nvmlDeviceEnumvs, 27
NVML_PSTATE_14
nvmlDeviceEnumvs, 27
NVML_PSTATE_15
nvmlDeviceEnumvs, 27
NVML_PSTATE_2
nvmlDeviceEnumvs, 27
NVML_PSTATE_3
nvmlDeviceEnumvs, 27
NVML_PSTATE_4
nvmlDeviceEnumvs, 27
NVML_PSTATE_5
nvmlDeviceEnumvs, 27
NVML_PSTATE_6
nvmlDeviceEnumvs, 27
NVML_PSTATE_7
nvmlDeviceEnumvs, 27
NVML_PSTATE_8
nvmlDeviceEnumvs, 27
NVML_PSTATE_9
nvmlDeviceEnumvs, 27
NVML_PSTATE_UNKNOWN
nvmlDeviceEnumvs, 27
NVML_SUCCESS
nvmlDeviceEnumvs, 27
NVML_TEMPERATURE_GPU
nvmlDeviceEnumvs, 28
NVML_VOLATILE_ECC
nvmlDeviceEnumvs, 25
NVML_DEVICE_INFOROM_VERSION_BUFFER_-
SIZE
nvmlConstants, 33
NVML_DEVICE_NAME_BUFFER_SIZE
nvmlConstants, 33
NVML_DEVICE_SERIAL_BUFFER_SIZE
nvmlConstants, 33
NVML_DEVICE_UUID_BUFFER_SIZE
nvmlConstants, 33
NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE
nvmlConstants, 33
NVML_DOUBLE_BIT_ECC
nvmlDeviceEnumvs, 24
NVML_SINGLE_BIT_ECC
nvmlDeviceEnumvs, 24
NVML_SYSTEM_DRIVER_VERSION_BUFFER_-
SIZE
nvmlConstants, 33
NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE
nvmlConstants, 33
NVML_VALUE_NOT_AVAILABLE
nvmlDeviceStructs, 21
nvmlClocksThrottleReasonAll
nvmlClocksThrottleReasons, 77
nvmlClocksThrottleReasonGpuIdle
nvmlClocksThrottleReasons, 77
nvmlClocksThrottleReasonHwSlowdown
nvmlClocksThrottleReasons, 77
nvmlClocksThrottleReasonNone
nvmlClocksThrottleReasons, 77
NvmlClocksThrottleReasons, 77
nvmlClocksThrottleReasons
nvmlClocksThrottleReasonAll, 77
nvmlClocksThrottleReasonGpuIdle, 77
nvmlClocksThrottleReasonHwSlowdown, 77
nvmlClocksThrottleReasonNone, 77
nvmlClocksThrottleReasonSwPowerCap, 78
nvmlClocksThrottleReasonUnknown, 78
nvmlClocksThrottleReasonUserDefinedClocks, 78
nvmlClocksThrottleReasonSwPowerCap
nvmlClocksThrottleReasons, 78
nvmlClocksThrottleReasonUnknown
nvmlClocksThrottleReasons, 78
nvmlClocksThrottleReasonUserDefinedClocks
nvmlClocksThrottleReasons, 78
nvmlClockType_t
nvmlDeviceEnumvs, 24
nvmlComputeMode_t
nvmlDeviceEnumvs, 24
nvmlConstants

NVML_DEVICE_INFOROM_VERSION_-
BUFFER_SIZE, 33
NVML_DEVICE_NAME_BUFFER_SIZE, 33
NVML_DEVICE_SERIAL_BUFFER_SIZE, 33
NVML_DEVICE_UUID_BUFFER_SIZE, 33
NVML_DEVICE_VBIOS_VERSION_BUFFER_-
SIZE, 33
NVML_SYSTEM_DRIVER_VERSION_-
BUFFER_SIZE, 33
NVML_SYSTEM_NVML_VERSION_BUFFER_-
SIZE, 33
nvmlDeviceClearEccErrorCounts
nvmlDeviceCommands, 67
nvmlDeviceCommands
nvmlDeviceClearEccErrorCounts, 67
nvmlDeviceSetApplicationsClocks, 68
nvmlDeviceSetComputeMode, 68
nvmlDeviceSetDriverModel, 69
nvmlDeviceSetEccMode, 70
nvmlDeviceSetGpuOperationMode, 70
nvmlDeviceSetPersistenceMode, 71
nvmlDeviceSetPowerManagementLimit, 71
nvmlDeviceEnumvs
NVML_AGGREGATE_ECC, 25
NVML_CLOCK_GRAPHICS, 24
NVML_CLOCK_MEM, 24
NVML_CLOCK_SM, 24
NVML_COMPUTEMODE_DEFAULT, 25
NVML_COMPUTEMODE_EXCLUSIVE_-
PROCESS, 25
NVML_COMPUTEMODE_EXCLUSIVE_-
THREAD, 25
NVML_COMPUTEMODE_PROHIBITED, 25
NVML_DRIVER_WDDM, 25
NVML_DRIVER_WDM, 25
NVML_ERROR_ALREADY_INITIALIZED, 27
NVML_ERROR_CORRUPTED_INFOROM, 27
NVML_ERROR_DRIVER_NOT_LOADED, 27
NVML_ERROR_FUNCTION_NOT_FOUND, 27
NVML_ERROR_INSUFFICIENT_POWER, 27
NVML_ERROR_INSUFFICIENT_SIZE, 27
NVML_ERROR_INVALID_ARGUMENT, 27
NVML_ERROR_IRQ_ISSUE, 27
NVML_ERROR_LIBRARY_NOT_FOUND, 27
NVML_ERROR_NO_PERMISSION, 27
NVML_ERROR_NOT_FOUND, 27
NVML_ERROR_NOT_SUPPORTED, 27
NVML_ERROR_TIMEOUT, 27
NVML_ERROR_UNINITIALIZED, 27
NVML_ERROR_UNKNOWN, 27
NVML_FEATURE_DISABLED, 25
NVML_FEATURE_ENABLED, 25
NVML_GOM_ALL_ON, 26
NVML_GOM_COMPUTE, 26
NVML_GOM_LOW_DP, 26
NVML_INFOROM_COUNT, 26
NVML_INFOROM_ECC, 26
NVML_INFOROM_OEM, 26
NVML_INFOROM_POWER, 26
NVML_MEMORY_ERROR_TYPE_-
CORRECTED, 26
NVML_MEMORY_ERROR_TYPE_COUNT, 26
NVML_MEMORY_ERROR_TYPE_-
UNCORRECTED, 26
NVML_MEMORY_LOCATION_COUNT, 26
NVML_MEMORY_LOCATION_DEVICE_-
MEMORY, 26
NVML_MEMORY_LOCATION_L1_CACHE, 26
NVML_MEMORY_LOCATION_L2_CACHE, 26
NVML_MEMORY_LOCATION_REGISTER_-
FILE, 26
NVML_MEMORY_LOCATION_TEXTURE_-
MEMORY, 26
NVML_PSTATE_0, 27
NVML_PSTATE_1, 27
NVML_PSTATE_10, 27
NVML_PSTATE_11, 27
NVML_PSTATE_12, 27
NVML_PSTATE_13, 27
NVML_PSTATE_14, 27
NVML_PSTATE_15, 27
NVML_PSTATE_2, 27
NVML_PSTATE_3, 27
NVML_PSTATE_4, 27
NVML_PSTATE_5, 27
NVML_PSTATE_6, 27
NVML_PSTATE_7, 27
NVML_PSTATE_8, 27
NVML_PSTATE_9, 27
NVML_PSTATE_UNKNOWN, 27
NVML_SUCCESS, 27
NVML_TEMPERATURE_GPU, 28
NVML_VOLATILE_ECC, 25
NVML_DOUBLE_BIT_ECC, 24
NVML_SINGLE_BIT_ECC, 24
nvmlClockType_t, 24
nvmlComputeMode_t, 24
nvmlDriverModel_t, 25
nvmlEccBitType_t, 24
nvmlEccCounterType_t, 25
nvmlEnableState_t, 25
nvmlGpuOperationMode_t, 25
nvmlInforomObject_t, 26
nvmlMemoryErrorType_t, 26
nvmlMemoryLocation_t, 26
nvmlPstates_t, 26
nvmlReturn_t, 27
nvmlTemperatureSensors_t, 27

nvmlDeviceGetApplicationsClock
nvmlDeviceQueries, 41
nvmlDeviceGetClockInfo
nvmlDeviceQueries, 42
nvmlDeviceGetComputeMode
nvmlDeviceQueries, 42
nvmlDeviceGetComputeRunningProcesses
nvmlDeviceQueries, 43
nvmlDeviceGetCount
nvmlDeviceQueries, 43
nvmlDeviceGetCurrentClocksThrottleReasons
nvmlDeviceQueries, 44
nvmlDeviceGetCurrPcieLinkGeneration
nvmlDeviceQueries, 44
nvmlDeviceGetCurrPcieLinkWidth
nvmlDeviceQueries, 44
nvmlDeviceGetDetailedEccErrors
nvmlDeviceQueries, 45
nvmlDeviceGetDisplayMode
nvmlDeviceQueries, 46
nvmlDeviceGetDriverModel
nvmlDeviceQueries, 46
nvmlDeviceGetEccMode
nvmlDeviceQueries, 47
nvmlDeviceGetFanSpeed
nvmlDeviceQueries, 47
nvmlDeviceGetGpuOperationMode
nvmlDeviceQueries, 48
nvmlDeviceGetHandleByIndex
nvmlDeviceQueries, 48
nvmlDeviceGetHandleByPciBusId
nvmlDeviceQueries, 49
nvmlDeviceGetHandleBySerial
nvmlDeviceQueries, 49
nvmlDeviceGetHandleByUUID
nvmlDeviceQueries, 50
nvmlDeviceGetInforomConfigurationChecksum
nvmlDeviceQueries, 50
nvmlDeviceGetInforomImageVersion
nvmlDeviceQueries, 51
nvmlDeviceGetInforomVersion
nvmlDeviceQueries, 51
nvmlDeviceGetMaxClockInfo
nvmlDeviceQueries, 52
nvmlDeviceGetMaxPcieLinkGeneration
nvmlDeviceQueries, 52
nvmlDeviceGetMaxPcieLinkWidth
nvmlDeviceQueries, 53
nvmlDeviceGetMemoryErrorCounter
nvmlDeviceQueries, 53
nvmlDeviceGetMemoryInfo
nvmlDeviceQueries, 54
nvmlDeviceGetName
nvmlDeviceQueries, 54
nvmlDeviceGetPciInfo
nvmlDeviceQueries, 55
nvmlDeviceGetPerformanceState
nvmlDeviceQueries, 55
nvmlDeviceGetPersistenceMode
nvmlDeviceQueries, 56
nvmlDeviceGetPowerManagementDefaultLimit
nvmlDeviceQueries, 56
nvmlDeviceGetPowerManagementLimit
nvmlDeviceQueries, 57
nvmlDeviceGetPowerManagementLimitConstraints
nvmlDeviceQueries, 57
nvmlDeviceGetPowerManagementMode
nvmlDeviceQueries, 58
nvmlDeviceGetPowerState
nvmlDeviceQueries, 58
nvmlDeviceGetPowerUsage
nvmlDeviceQueries, 59
nvmlDeviceGetSerial
nvmlDeviceQueries, 59
nvmlDeviceGetSupportedClocksThrottleReasons
nvmlDeviceQueries, 60
nvmlDeviceGetSupportedEventTypes
nvmlEvents, 73
nvmlDeviceGetSupportedGraphicsClocks
nvmlDeviceQueries, 60
nvmlDeviceGetSupportedMemoryClocks
nvmlDeviceQueries, 61
nvmlDeviceGetTemperature
nvmlDeviceQueries, 62
nvmlDeviceGetTotalEccErrors
nvmlDeviceQueries, 62
nvmlDeviceGetUtilizationRates
nvmlDeviceQueries, 63
nvmlDeviceGetUUID
nvmlDeviceQueries, 63
nvmlDeviceGetVbiosVersion
nvmlDeviceQueries, 64
nvmlDeviceOnSameBoard
nvmlDeviceQueries, 64
nvmlDeviceQueries
nvmlDeviceGetApplicationsClock, 41
nvmlDeviceGetClockInfo, 42
nvmlDeviceGetComputeMode, 42
nvmlDeviceGetComputeRunningProcesses, 43
nvmlDeviceGetCount, 43
nvmlDeviceGetCurrentClocksThrottleReasons, 44
nvmlDeviceGetCurrPcieLinkGeneration, 44
nvmlDeviceGetCurrPcieLinkWidth, 44
nvmlDeviceGetDetailedEccErrors, 45
nvmlDeviceGetDisplayMode, 46
nvmlDeviceGetDriverModel, 46
nvmlDeviceGetEccMode, 47
nvmlDeviceGetFanSpeed, 47

nvmlDeviceGetGpuOperationMode, 48
nvmlDeviceGetHandleByIndex, 48
nvmlDeviceGetHandleByPciBusId, 49
nvmlDeviceGetHandleBySerial, 49
nvmlDeviceGetHandleByUUID, 50
nvmlDeviceGetInforomConfigurationChecksum, 50
nvmlDeviceGetInforomImageVersion, 51
nvmlDeviceGetInforomVersion, 51
nvmlDeviceGetMaxClockInfo, 52
nvmlDeviceGetMaxPcieLinkGeneration, 52
nvmlDeviceGetMaxPcieLinkWidth, 53
nvmlDeviceGetMemoryErrorCounter, 53
nvmlDeviceGetMemoryInfo, 54
nvmlDeviceGetName, 54
nvmlDeviceGetPciInfo, 55
nvmlDeviceGetPerformanceState, 55
nvmlDeviceGetPersistenceMode, 56
nvmlDeviceGetPowerManagementDefaultLimit, 56
nvmlDeviceGetPowerManagementLimit, 57
nvmlDeviceGetPowerManagementLimitConstraints, 57
nvmlDeviceGetPowerManagementMode, 58
nvmlDeviceGetPowerState, 58
nvmlDeviceGetPowerUsage, 59
nvmlDeviceGetSerial, 59
nvmlDeviceGetSupportedClocksThrottleReasons, 60
nvmlDeviceGetSupportedGraphicsClocks, 60
nvmlDeviceGetSupportedMemoryClocks, 61
nvmlDeviceGetTemperature, 62
nvmlDeviceGetTotalEccErrors, 62
nvmlDeviceGetUtilizationRates, 63
nvmlDeviceGetUUID, 63
nvmlDeviceGetVbiosVersion, 64
nvmlDeviceOnSameBoard, 64
nvmlDeviceResetApplicationsClocks, 64
nvmlDeviceValidateInforom, 65
nvmlDeviceRegisterEvents
nvmlEvents, 74
nvmlDeviceResetApplicationsClocks
nvmlDeviceQueries, 64
nvmlDeviceSetApplicationsClocks
nvmlDeviceCommands, 68
nvmlDeviceSetComputeMode
nvmlDeviceCommands, 68
nvmlDeviceSetDriverModel
nvmlDeviceCommands, 69
nvmlDeviceSetEccMode
nvmlDeviceCommands, 70
nvmlDeviceSetGpuOperationMode
nvmlDeviceCommands, 70
nvmlDeviceSetPersistenceMode
nvmlDeviceCommands, 71
nvmlDeviceSetPowerManagementLimit
nvmlDeviceCommands, 71
nvmlDeviceStructs
NVML_VALUE_NOT_AVAILABLE, 21
nvmlDeviceValidateInforom
nvmlDeviceQueries, 65
nvmlDriverModel_t
nvmlDeviceEnumvs, 25
nvmlEccBitType_t
nvmlDeviceEnumvs, 24
nvmlEccCounterType_t
nvmlDeviceEnumvs, 25
nvmlEccErrorCounts_t, 79
nvmlEnableState_t
nvmlDeviceEnumvs, 25
nvmlErrorReporting
nvmlErrorString, 32
nvmlErrorString
nvmlErrorReporting, 32
nvmlEventData_t, 80
nvmlEvents
nvmlDeviceGetSupportedEventTypes, 73
nvmlDeviceRegisterEvents, 74
nvmlEventSet_t, 73
nvmlEventSetCreate, 74
nvmlEventSetFree, 75
nvmlEventSetWait, 75
nvmlEventSet_t
nvmlEvents, 73
nvmlEventSetCreate
nvmlEvents, 74
nvmlEventSetFree
nvmlEvents, 75
nvmlEventSetWait
nvmlEvents, 75
nvmlEventType
nvmlEventTypeClock, 30
nvmlEventTypePState, 30
nvmlEventTypeClock
nvmlEventType, 30
nvmlEventTypePState
nvmlEventType, 30
nvmlFanState_t
nvmlUnitStructs, 29
nvmlGpuOperationMode_t
nvmlDeviceEnumvs, 25
nvmlHwbcEntry_t, 81
nvmlInforomObject_t
nvmlDeviceEnumvs, 26
nvmlInit
nvmlInitializationAndCleanup, 31
nvmlInitializationAndCleanup
nvmlInit, 31
nvmlShutdown, 31
nvmlLedColor_t

nvmlUnitStructs, 29
nvmlLedState_t, 82
nvmlMemory_t, 83
nvmlMemoryErrorType_t
nvmlDeviceEnumvs, 26
nvmlMemoryLocation_t
nvmlDeviceEnumvs, 26
nvmlPciInfo_t, 84
nvmlProcessInfo_t, 85
nvmlPstates_t
nvmlDeviceEnumvs, 26
nvmlPSUInfo_t, 86
nvmlReturn_t
nvmlDeviceEnumvs, 27
nvmlShutdown
nvmlInitializationAndCleanup, 31
nvmlSystemGetDriverVersion
nvmlSystemQueries, 34
nvmlSystemGetHicVersion
nvmlUnitQueries, 36
nvmlSystemGetNVMLVersion
nvmlSystemQueries, 34
nvmlSystemGetProcessName
nvmlSystemQueries, 35
nvmlSystemQueries
nvmlSystemGetDriverVersion, 34
nvmlSystemGetNVMLVersion, 34
nvmlSystemGetProcessName, 35
nvmlTemperatureSensors_t
nvmlDeviceEnumvs, 27
nvmlUnitCommands
nvmlUnitSetLedState, 66
nvmlUnitFanInfo_t, 87
nvmlUnitFanSpeeds_t, 88
nvmlUnitGetCount
nvmlUnitQueries, 36
nvmlUnitGetDevices
nvmlUnitQueries, 37
nvmlUnitGetFanSpeedInfo
nvmlUnitQueries, 37
nvmlUnitGetHandleByIndex
nvmlUnitQueries, 37
nvmlUnitGetLedState
nvmlUnitQueries, 38
nvmlUnitGetPsuInfo
nvmlUnitQueries, 38
nvmlUnitGetTemperature
nvmlUnitQueries, 39
nvmlUnitGetUnitInfo
nvmlUnitQueries, 39
nvmlUnitInfo_t, 89
nvmlUnitQueries
nvmlSystemGetHicVersion, 36
nvmlUnitGetCount, 36
nvmlUnitGetDevices, 37
nvmlUnitGetFanSpeedInfo, 37
nvmlUnitGetHandleByIndex, 37
nvmlUnitGetLedState, 38
nvmlUnitGetPsuInfo, 38
nvmlUnitGetTemperature, 39
nvmlUnitGetUnitInfo, 39
nvmlUnitSetLedState
nvmlUnitCommands, 66
nvmlUnitStructs
NVML_FAN_FAILED, 29
NVML_FAN_NORMAL, 29
NVML_LED_COLOR_AMBER, 29
NVML_LED_COLOR_GREEN, 29
nvmlFanState_t, 29
nvmlLedColor_t, 29
nvmlUtilization_t, 90

System Queries, 34

Unit Commands, 66
Unit Queries, 36
Unit Structs, 29

Notice

Trademarks

Copyright