



# NVML API REFERENCE MANUAL

September 8, 2016

Version 361.94





# Contents

<b>1</b>	<b>Known issues in the current version of NVML library</b>	<b>1</b>
<b>2</b>	<b>Change log of NVML library</b>	<b>3</b>
2.1	Changes between NVML v346 Update and v349 . . . . .	4
2.2	Changes between NVML v340 Update and v346 . . . . .	4
2.3	Changes between NVML v331 Update and v340 . . . . .	4
2.4	Changes between NVML v5.319 Update and v331 . . . . .	5
2.5	Changes between NVML v5.319 RC and v5.319 Update . . . . .	5
2.6	Changes between NVML v4.304 and v5.319 RC . . . . .	5
2.7	Changes between NVML v4.304 RC and v4.304 Production . . . . .	6
2.8	Changes between NVML v3.295 and v4.304 RC . . . . .	6
2.9	Changes between NVML v2.285 and v3.295 . . . . .	7
2.10	Changes between NVML v1.0 and v2.285 . . . . .	7
<b>3</b>	<b>Deprecated List</b>	<b>9</b>
<b>4</b>	<b>Module Index</b>	<b>11</b>
4.1	Modules . . . . .	11
<b>5</b>	<b>Data Structure Index</b>	<b>13</b>
5.1	Data Structures . . . . .	13
<b>6</b>	<b>Module Documentation</b>	<b>15</b>
6.1	Device Structs . . . . .	15
6.1.1	Define Documentation . . . . .	16
6.1.1.1	NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE . . . . .	16
6.1.1.2	NVML_MAX_PHYSICAL_BRIDGE . . . . .	16
6.1.1.3	NVML_NVLINK_MAX_LINKS . . . . .	16
6.1.1.4	NVML_VALUE_NOT_AVAILABLE . . . . .	16
6.1.2	Enumeration Type Documentation . . . . .	16

6.1.2.1	<a href="#">nvmlBridgeChipType_t</a>	16
6.1.2.2	<a href="#">nvmlGpuTopologyLevel_t</a>	16
6.1.2.3	<a href="#">nvmlNvLinkCapability_t</a>	16
6.1.2.4	<a href="#">nvmlNvLinkErrorCounter_t</a>	17
6.1.2.5	<a href="#">nvmlNvLinkUtilizationCountPktTypes_t</a>	17
6.1.2.6	<a href="#">nvmlNvLinkUtilizationCountUnits_t</a>	17
6.1.2.7	<a href="#">nvmlPcieUtilCounter_t</a>	17
6.1.2.8	<a href="#">nvmlPerfPolicyType_t</a>	17
6.1.2.9	<a href="#">nvmlSamplingType_t</a>	17
6.1.2.10	<a href="#">nvmlValueType_t</a>	17
6.2	<a href="#">Device Enums</a>	18
6.2.1	<a href="#">Define Documentation</a>	20
6.2.1.1	<a href="#">NVML_DOUBLE_BIT_ECC</a>	20
6.2.1.2	<a href="#">NVML_SINGLE_BIT_ECC</a>	20
6.2.1.3	<a href="#">nvmlEccBitType_t</a>	21
6.2.2	<a href="#">Enumeration Type Documentation</a>	21
6.2.2.1	<a href="#">nvmlBrandType_t</a>	21
6.2.2.2	<a href="#">nvmlClockId_t</a>	21
6.2.2.3	<a href="#">nvmlClockType_t</a>	21
6.2.2.4	<a href="#">nvmlComputeMode_t</a>	21
6.2.2.5	<a href="#">nvmlDriverModel_t</a>	22
6.2.2.6	<a href="#">nvmlEccCounterType_t</a>	22
6.2.2.7	<a href="#">nvmlEnableState_t</a>	22
6.2.2.8	<a href="#">nvmlGpuOperationMode_t</a>	22
6.2.2.9	<a href="#">nvmlInforomObject_t</a>	23
6.2.2.10	<a href="#">nvmlMemoryErrorType_t</a>	23
6.2.2.11	<a href="#">nvmlMemoryLocation_t</a>	23
6.2.2.12	<a href="#">nvmlPageRetirementCause_t</a>	23
6.2.2.13	<a href="#">nvmlPstates_t</a>	24
6.2.2.14	<a href="#">nvmlRestrictedAPI_t</a>	24
6.2.2.15	<a href="#">nvmlReturn_t</a>	24
6.2.2.16	<a href="#">nvmlTemperatureSensors_t</a>	25
6.2.2.17	<a href="#">nvmlTemperatureThresholds_t</a>	25
6.3	<a href="#">Unit Structs</a>	26
6.3.1	<a href="#">Enumeration Type Documentation</a>	26
6.3.1.1	<a href="#">nvmlFanState_t</a>	26
6.3.1.2	<a href="#">nvmlLedColor_t</a>	26

6.4	Event Types	27
6.4.1	Detailed Description	27
6.4.2	Define Documentation	27
6.4.2.1	<code>nvmlEventTypeClock</code>	27
6.4.2.2	<code>nvmlEventTypeDoubleBitEccError</code>	27
6.4.2.3	<code>nvmlEventTypePState</code>	28
6.4.2.4	<code>nvmlEventTypeSingleBitEccError</code>	28
6.5	Accounting Statistics	29
6.5.1	Detailed Description	29
6.5.2	Function Documentation	29
6.5.2.1	<code>nvmlDeviceClearAccountingPids</code>	29
6.5.2.2	<code>nvmlDeviceGetAccountingBufferSize</code>	30
6.5.2.3	<code>nvmlDeviceGetAccountingMode</code>	30
6.5.2.4	<code>nvmlDeviceGetAccountingPids</code>	31
6.5.2.5	<code>nvmlDeviceGetAccountingStats</code>	31
6.5.2.6	<code>nvmlDeviceSetAccountingMode</code>	32
6.6	Initialization and Cleanup	33
6.6.1	Detailed Description	33
6.6.2	Function Documentation	33
6.6.2.1	<code>nvmlInit</code>	33
6.6.2.2	<code>nvmlShutdown</code>	33
6.7	Error reporting	35
6.7.1	Detailed Description	35
6.7.2	Function Documentation	35
6.7.2.1	<code>nvmlErrorString</code>	35
6.8	Constants	36
6.8.1	Define Documentation	36
6.8.1.1	<code>NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE</code>	36
6.8.1.2	<code>NVML_DEVICE_NAME_BUFFER_SIZE</code>	36
6.8.1.3	<code>NVML_DEVICE_PART_NUMBER_BUFFER_SIZE</code>	36
6.8.1.4	<code>NVML_DEVICE_SERIAL_BUFFER_SIZE</code>	36
6.8.1.5	<code>NVML_DEVICE_UUID_BUFFER_SIZE</code>	36
6.8.1.6	<code>NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE</code>	36
6.8.1.7	<code>NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE</code>	36
6.8.1.8	<code>NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE</code>	36
6.9	System Queries	37
6.9.1	Detailed Description	37

6.9.2	Function Documentation	37
6.9.2.1	nvmlSystemGetDriverVersion	37
6.9.2.2	nvmlSystemGetNVMLVersion	37
6.9.2.3	nvmlSystemGetProcessName	38
6.10	Unit Queries	39
6.10.1	Detailed Description	39
6.10.2	Function Documentation	39
6.10.2.1	nvmlSystemGetHicVersion	39
6.10.2.2	nvmlUnitGetCount	39
6.10.2.3	nvmlUnitGetDevices	40
6.10.2.4	nvmlUnitGetFanSpeedInfo	40
6.10.2.5	nvmlUnitGetHandleByIndex	41
6.10.2.6	nvmlUnitGetLedState	41
6.10.2.7	nvmlUnitGetPsuInfo	41
6.10.2.8	nvmlUnitGetTemperature	42
6.10.2.9	nvmlUnitGetUnitInfo	42
6.11	Device Queries	43
6.11.1	Detailed Description	45
6.11.2	Function Documentation	45
6.11.2.1	nvmlDeviceClearCpuAffinity	45
6.11.2.2	nvmlDeviceGetAPIRestriction	46
6.11.2.3	nvmlDeviceGetApplicationsClock	46
6.11.2.4	nvmlDeviceGetAutoBoostedClocksEnabled	47
6.11.2.5	nvmlDeviceGetBAR1MemoryInfo	47
6.11.2.6	nvmlDeviceGetBoardId	48
6.11.2.7	nvmlDeviceGetBoardPartNumber	48
6.11.2.8	nvmlDeviceGetBrand	49
6.11.2.9	nvmlDeviceGetBridgeChipInfo	49
6.11.2.10	nvmlDeviceGetClock	50
6.11.2.11	nvmlDeviceGetClockInfo	50
6.11.2.12	nvmlDeviceGetComputeMode	51
6.11.2.13	nvmlDeviceGetComputeRunningProcesses	51
6.11.2.14	nvmlDeviceGetCount	52
6.11.2.15	nvmlDeviceGetCpuAffinity	52
6.11.2.16	nvmlDeviceGetCurrentClocksThrottleReasons	53
6.11.2.17	nvmlDeviceGetCurrPcieLinkGeneration	53
6.11.2.18	nvmlDeviceGetCurrPcieLinkWidth	54

6.11.2.19 nvmlDeviceGetDecoderUtilization . . . . .	54
6.11.2.20 nvmlDeviceGetDefaultApplicationsClock . . . . .	55
6.11.2.21 nvmlDeviceGetDetailedEccErrors . . . . .	55
6.11.2.22 nvmlDeviceGetDisplayActive . . . . .	56
6.11.2.23 nvmlDeviceGetDisplayMode . . . . .	56
6.11.2.24 nvmlDeviceGetDriverModel . . . . .	57
6.11.2.25 nvmlDeviceGetEccMode . . . . .	58
6.11.2.26 nvmlDeviceGetEncoderUtilization . . . . .	58
6.11.2.27 nvmlDeviceGetEnforcedPowerLimit . . . . .	59
6.11.2.28 nvmlDeviceGetFanSpeed . . . . .	59
6.11.2.29 nvmlDeviceGetGpuOperationMode . . . . .	60
6.11.2.30 nvmlDeviceGetGraphicsRunningProcesses . . . . .	60
6.11.2.31 nvmlDeviceGetHandleByIndex . . . . .	61
6.11.2.32 nvmlDeviceGetHandleByPciBusId . . . . .	62
6.11.2.33 nvmlDeviceGetHandleBySerial . . . . .	62
6.11.2.34 nvmlDeviceGetHandleByUUID . . . . .	63
6.11.2.35 nvmlDeviceGetIndex . . . . .	64
6.11.2.36 nvmlDeviceGetInforomConfigurationChecksum . . . . .	64
6.11.2.37 nvmlDeviceGetInforomImageVersion . . . . .	65
6.11.2.38 nvmlDeviceGetInforomVersion . . . . .	66
6.11.2.39 nvmlDeviceGetMaxClockInfo . . . . .	66
6.11.2.40 nvmlDeviceGetMaxCustomerBoostClock . . . . .	67
6.11.2.41 nvmlDeviceGetMaxPcieLinkGeneration . . . . .	67
6.11.2.42 nvmlDeviceGetMaxPcieLinkWidth . . . . .	68
6.11.2.43 nvmlDeviceGetMemoryErrorCounter . . . . .	68
6.11.2.44 nvmlDeviceGetMemoryInfo . . . . .	69
6.11.2.45 nvmlDeviceGetMinorNumber . . . . .	69
6.11.2.46 nvmlDeviceGetMultiGpuBoard . . . . .	70
6.11.2.47 nvmlDeviceGetName . . . . .	70
6.11.2.48 nvmlDeviceGetPcieReplayCounter . . . . .	71
6.11.2.49 nvmlDeviceGetPcieThroughput . . . . .	71
6.11.2.50 nvmlDeviceGetPciInfo . . . . .	72
6.11.2.51 nvmlDeviceGetPerformanceState . . . . .	72
6.11.2.52 nvmlDeviceGetPersistenceMode . . . . .	73
6.11.2.53 nvmlDeviceGetPowerManagementDefaultLimit . . . . .	73
6.11.2.54 nvmlDeviceGetPowerManagementLimit . . . . .	74
6.11.2.55 nvmlDeviceGetPowerManagementLimitConstraints . . . . .	74

6.11.2.56 nvmlDeviceGetPowerManagementMode . . . . .	75
6.11.2.57 nvmlDeviceGetPowerState . . . . .	75
6.11.2.58 nvmlDeviceGetPowerUsage . . . . .	76
6.11.2.59 nvmlDeviceGetRetiredPages . . . . .	76
6.11.2.60 nvmlDeviceGetRetiredPagesPendingStatus . . . . .	77
6.11.2.61 nvmlDeviceGetSamples . . . . .	77
6.11.2.62 nvmlDeviceGetSerial . . . . .	78
6.11.2.63 nvmlDeviceGetSupportedClocksThrottleReasons . . . . .	79
6.11.2.64 nvmlDeviceGetSupportedGraphicsClocks . . . . .	79
6.11.2.65 nvmlDeviceGetSupportedMemoryClocks . . . . .	80
6.11.2.66 nvmlDeviceGetTemperature . . . . .	80
6.11.2.67 nvmlDeviceGetTemperatureThreshold . . . . .	81
6.11.2.68 nvmlDeviceGetTopologyCommonAncestor . . . . .	81
6.11.2.69 nvmlDeviceGetTopologyNearestGpus . . . . .	82
6.11.2.70 nvmlDeviceGetTotalEccErrors . . . . .	82
6.11.2.71 nvmlDeviceGetUtilizationRates . . . . .	83
6.11.2.72 nvmlDeviceGetUUID . . . . .	83
6.11.2.73 nvmlDeviceGetVbiosVersion . . . . .	84
6.11.2.74 nvmlDeviceGetViolationStatus . . . . .	84
6.11.2.75 nvmlDeviceOnSameBoard . . . . .	85
6.11.2.76 nvmlDeviceResetApplicationsClocks . . . . .	85
6.11.2.77 nvmlDeviceSetAutoBoostedClocksEnabled . . . . .	86
6.11.2.78 nvmlDeviceSetCpuAffinity . . . . .	86
6.11.2.79 nvmlDeviceSetDefaultAutoBoostedClocksEnabled . . . . .	87
6.11.2.80 nvmlDeviceValidateInforom . . . . .	87
6.11.2.81 nvmlSystemGetTopologyGpuSet . . . . .	88
6.12 Unit Commands . . . . .	89
6.12.1 Detailed Description . . . . .	89
6.12.2 Function Documentation . . . . .	89
6.12.2.1 nvmlUnitSetLedState . . . . .	89
6.13 Device Commands . . . . .	90
6.13.1 Detailed Description . . . . .	90
6.13.2 Function Documentation . . . . .	90
6.13.2.1 nvmlDeviceClearEccErrorCounts . . . . .	90
6.13.2.2 nvmlDeviceSetAPIRestriction . . . . .	91
6.13.2.3 nvmlDeviceSetApplicationsClocks . . . . .	91
6.13.2.4 nvmlDeviceSetComputeMode . . . . .	92



6.13.2.5	<a href="#">nvmlDeviceSetDriverModel</a>	93
6.13.2.6	<a href="#">nvmlDeviceSetEccMode</a>	94
6.13.2.7	<a href="#">nvmlDeviceSetGpuOperationMode</a>	94
6.13.2.8	<a href="#">nvmlDeviceSetPersistenceMode</a>	95
6.13.2.9	<a href="#">nvmlDeviceSetPowerManagementLimit</a>	95
6.14	<a href="#">NvLink Methods</a>	97
6.14.1	<a href="#">Detailed Description</a>	97
6.14.2	<a href="#">Function Documentation</a>	97
6.14.2.1	<a href="#">nvmlDeviceFreezeNvLinkUtilizationCounter</a>	97
6.14.2.2	<a href="#">nvmlDeviceGetNvLinkCapability</a>	98
6.14.2.3	<a href="#">nvmlDeviceGetNvLinkErrorCounter</a>	98
6.14.2.4	<a href="#">nvmlDeviceGetNvLinkRemotePciInfo</a>	99
6.14.2.5	<a href="#">nvmlDeviceGetNvLinkState</a>	99
6.14.2.6	<a href="#">nvmlDeviceGetNvLinkUtilizationControl</a>	99
6.14.2.7	<a href="#">nvmlDeviceGetNvLinkUtilizationCounter</a>	100
6.14.2.8	<a href="#">nvmlDeviceGetNvLinkVersion</a>	100
6.14.2.9	<a href="#">nvmlDeviceResetNvLinkErrorCounters</a>	101
6.14.2.10	<a href="#">nvmlDeviceResetNvLinkUtilizationCounter</a>	101
6.14.2.11	<a href="#">nvmlDeviceSetNvLinkUtilizationControl</a>	102
6.15	<a href="#">Event Handling Methods</a>	103
6.15.1	<a href="#">Detailed Description</a>	103
6.15.2	<a href="#">Typedef Documentation</a>	103
6.15.2.1	<a href="#">nvmlEventSet_t</a>	103
6.15.3	<a href="#">Function Documentation</a>	103
6.15.3.1	<a href="#">nvmlDeviceGetSupportedEventTypes</a>	103
6.15.3.2	<a href="#">nvmlDeviceRegisterEvents</a>	104
6.15.3.3	<a href="#">nvmlEventSetCreate</a>	105
6.15.3.4	<a href="#">nvmlEventSetFree</a>	105
6.15.3.5	<a href="#">nvmlEventSetWait</a>	105
6.16	<a href="#">Drain states</a>	107
6.16.1	<a href="#">Detailed Description</a>	107
6.16.2	<a href="#">Function Documentation</a>	107
6.16.2.1	<a href="#">nvmlDeviceDiscoverGpus</a>	107
6.16.2.2	<a href="#">nvmlDeviceModifyDrainState</a>	107
6.16.2.3	<a href="#">nvmlDeviceQueryDrainState</a>	108
6.16.2.4	<a href="#">nvmlDeviceRemoveGpu</a>	108
6.17	<a href="#">NvmlClocksThrottleReasons</a>	110

6.17.1	Define Documentation	110
6.17.1.1	nvmlClocksThrottleReasonAll	110
6.17.1.2	nvmlClocksThrottleReasonApplicationsClocksSetting	110
6.17.1.3	nvmlClocksThrottleReasonGpuIdle	110
6.17.1.4	nvmlClocksThrottleReasonHwSlowdown	111
6.17.1.5	nvmlClocksThrottleReasonNone	111
6.17.1.6	nvmlClocksThrottleReasonSwPowerCap	111
6.17.1.7	nvmlClocksThrottleReasonSyncBoost	111
6.17.1.8	nvmlClocksThrottleReasonUnknown	111
6.17.1.9	nvmlClocksThrottleReasonUserDefinedClocks	112
<b>7</b>	<b>Data Structure Documentation</b>	<b>113</b>
7.1	nvmlAccountingStats_t Struct Reference	113
7.1.1	Detailed Description	114
7.2	nvmlBAR1Memory_t Struct Reference	115
7.2.1	Detailed Description	115
7.3	nvmlBridgeChipHierarchy_t Struct Reference	116
7.3.1	Detailed Description	116
7.4	nvmlBridgeChipInfo_t Struct Reference	117
7.4.1	Detailed Description	117
7.5	nvmlEccErrorCounts_t Struct Reference	118
7.5.1	Detailed Description	118
7.6	nvmlEventData_t Struct Reference	119
7.6.1	Detailed Description	119
7.7	nvmlHwbcEntry_t Struct Reference	120
7.7.1	Detailed Description	120
7.8	nvmlLedState_t Struct Reference	121
7.8.1	Detailed Description	121
7.9	nvmlMemory_t Struct Reference	122
7.9.1	Detailed Description	122
7.10	nvmlNvLinkUtilizationControl_t Struct Reference	123
7.10.1	Detailed Description	123
7.11	nvmlPciInfo_t Struct Reference	124
7.11.1	Detailed Description	124
7.12	nvmlProcessInfo_t Struct Reference	125
7.12.1	Detailed Description	125
7.13	nvmlPSUInfo_t Struct Reference	126

7.13.1 Detailed Description . . . . .	126
7.14 nvmlSample_t Struct Reference . . . . .	127
7.14.1 Detailed Description . . . . .	127
7.15 nvmlUnitFanInfo_t Struct Reference . . . . .	128
7.15.1 Detailed Description . . . . .	128
7.16 nvmlUnitFanSpeeds_t Struct Reference . . . . .	129
7.16.1 Detailed Description . . . . .	129
7.17 nvmlUnitInfo_t Struct Reference . . . . .	130
7.17.1 Detailed Description . . . . .	130
7.18 nvmlUtilization_t Struct Reference . . . . .	131
7.18.1 Detailed Description . . . . .	131
7.19 nvmlValue_t Union Reference . . . . .	132
7.19.1 Detailed Description . . . . .	132
7.20 nvmlViolationTime_t Struct Reference . . . . .	133
7.20.1 Detailed Description . . . . .	133



# Chapter 1

## Known issues in the current version of NVML library

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

- 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.
- [Accounting Statistics](#) supports only one process per GPU at a time (CUDA proxy server counts as one process).
- [nvmlAccountingStats\\_t::time](#) reports time and utilization values starting from cuInit till process termination. Next driver versions might change this behavior slightly and account process only from cuCtxCreate till cuCtxDestroy.
- On GPUs from Fermi family current P0 clocks (reported by [nvmlDeviceGetClockInfo](#)) can differ from max clocks by few MHz.



## **Chapter 2**

### **Change log of NVML library**

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

## 2.1 Changes between NVML v346 Update and v349

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

- Updated [nvmlDeviceGetMemoryInfo](#) to report Used/Free memory under Windows WDDM mode
- Added [nvmlDeviceGetTopologyCommonAncestor](#) to find the common path between two devices
- Added [nvmlDeviceGetTopologyNearestGpus](#) to get a set of GPUs given a path level
- Added [nvmlSystemGetTopologyGpuSet](#) to retrieve a set of GPUs with a given CPU affinity
- Updated [nvmlDeviceGetAccountingPids](#), [nvmlDeviceGetAccountingBufferSize](#) and [nvmlDeviceGetAccountingStats](#) to report accounting information for both active and terminated processes. The execution time field in [nvmlAccountingStats\\_t](#) structure is populated only when the process is terminated.

## 2.2 Changes between NVML v340 Update and v346

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

- added the public APIs [nvmlDeviceGetPcieReplayCounter](#) and [nvmlDeviceGetPcieThroughput](#)
- Discontinued Perl bindings support
- Added [nvmlDeviceGetGraphicsRunningProcesses](#) to get information about Graphics processes running on a GPU.

## 2.3 Changes between NVML v331 Update and v340

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

- Added [nvmlDeviceGetSamples](#) to get recent power, utilization and clock samples for the GPU.
- Added [nvmlDeviceGetTemperatureThreshold](#) to retrieve temperature threshold information.
- Added [nvmlDeviceGetBrand](#) to retrieve brand information (e.g. Tesla, Quadro, etc.)
- Added support for K40d and K80
- Added [nvmlDeviceGetTopology](#) internal API to retrieve path info between PCI devices (remove this for DITA)
- Added [nvmlDeviceGetViolationStatus](#) to get the duration of time during which the device was throttled (lower than requested clocks) due to thermal or power constraints.
- Added [nvmlDeviceGetEncoderUtilization](#) and [nvmlDeviceGetDecoderUtilization](#) APIs
- Added [nvmlDeviceGetCpuAffinity](#) to determine the closest processor(s) affinity to a specific GPU
- Added [nvmlDeviceSetCpuAffinity](#) to bind a specific GPU to the closest processor
- Added [nvmlDeviceClearCpuAffinity](#) to unbind a specific GPU
- Added [nvmlDeviceGetBoardId](#) to get a unique boardId for the running system



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

## 2.4 Changes between NVML v5.319 Update and v331

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

- Added [nvmlDeviceGetMinorNumber](#) to get the minor number for the device.
- Added [nvmlDeviceGetBAR1MemoryInfo](#) to get BAR1 total, available and used memory size.
- Added [nvmlDeviceGetBridgeChipInfo](#) to get the information related to bridge chip firmware.
- Added enforced power limit query API [nvmlDeviceGetEnforcedPowerLimit](#)
- Updated [nvmlEventSetWait](#) to return xid event data in case of xid error event.
- Added support for K8

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

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

- Added [nvmlDeviceSetAPIRestriction](#) and [nvmlDeviceGetAPIRestriction](#), with initial ability to toggle root-only requirement for [nvmlDeviceSetApplicationsClocks](#) and [nvmlDeviceResetApplicationsClocks](#).

## 2.6 Changes between NVML v4.304 and v5.319 RC

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

- IMPORTANT: Added \_v2 versions of [nvmlDeviceGetHandleByIndex](#) and [nvmlDeviceGetCount](#) that also count devices not accessible by current user
  - IMPORTANT: [nvmlDeviceGetHandleByIndex\\_v2](#) (default) can also return NVML\_ERROR\_NO\_PERMISSION
- Added [nvmlInit\\_v2](#) and [nvmlDeviceGetHandleByIndex\\_v2](#) that is safer and thus recommended function for initializing the library
  - [nvmlInit\\_v2](#) lazily initializes only requested devices (queried with [nvmlDeviceGetHandle\\*](#))
  - [nvml.h](#) defines [nvmlInit\\_v2](#) and [nvmlDeviceGetHandleByIndex\\_v2](#) as default functions
- Added [nvmlDeviceGetIndex](#)
- Added [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) to report GPUs that have fallen off the bus.
  - Note: All NVML device APIs can return this error code, as a GPU can fall off the bus at any time.

- Added new class of APIs for gathering process statistics ([Accounting Statistics](#))
- Application Clocks are no longer supported on GPU's from Quadro product line
- Added APIs to support dynamic page retirement. See [nvmlDeviceGetRetiredPages](#) and [nvmlDeviceGetRetiredPagesPendingStatus](#)
- Renamed `nvmlClocksThrottleReasonUserDefinedClocks` to `nvmlClocksThrottleReasonApplicationsClocksSetting`. Old name is deprecated and can be removed in one of the next major releases.
- Added [nvmlDeviceGetDisplayActive](#) and updated documentation to clarify how it differs from [nvmlDeviceGetDisplayMode](#)

## 2.7 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](#)

## 2.8 Changes between NVML v3.295 and v4.304 RC

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

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

## 2.9 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

## 2.10 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 3**

### **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](#).

**Global [nvmlClocksThrottleReasonUserDefinedClocks](#)** Renamed to [nvmlClocksThrottleReasonApplicationsClocksSetting](#) as the name describes the situation more accurately.

# Chapter 4

## Module Index

### 4.1 Modules

Here is a list of all modules:

Device Structs . . . . .	15
Device Enums . . . . .	18
Unit Structs . . . . .	26
Accounting Statistics . . . . .	29
Initialization and Cleanup . . . . .	33
Error reporting . . . . .	35
Constants . . . . .	36
System Queries . . . . .	37
Unit Queries . . . . .	39
Device Queries . . . . .	43
Unit Commands . . . . .	89
Device Commands . . . . .	90
NvLink Methods . . . . .	97
Event Handling Methods . . . . .	103
Event Types . . . . .	27
Drain states . . . . .	107
NvmlClocksThrottleReasons . . . . .	110





## Chapter 5

# Data Structure Index

### 5.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">nvmlAccountingStats_t</a>	113
<a href="#">nvmlBAR1Memory_t</a>	115
<a href="#">nvmlBridgeChipHierarchy_t</a>	116
<a href="#">nvmlBridgeChipInfo_t</a>	117
<a href="#">nvmlEccErrorCounts_t</a>	118
<a href="#">nvmlEventData_t</a>	119
<a href="#">nvmlHwbcEntry_t</a>	120
<a href="#">nvmlLedState_t</a>	121
<a href="#">nvmlMemory_t</a>	122
<a href="#">nvmlNvLinkUtilizationControl_t</a>	123
<a href="#">nvmlPciInfo_t</a>	124
<a href="#">nvmlProcessInfo_t</a>	125
<a href="#">nvmlPSUInfo_t</a>	126
<a href="#">nvmlSample_t</a>	127
<a href="#">nvmlUnitFanInfo_t</a>	128
<a href="#">nvmlUnitFanSpeeds_t</a>	129
<a href="#">nvmlUnitInfo_t</a>	130
<a href="#">nvmlUtilization_t</a>	131
<a href="#">nvmlValue_t</a>	132
<a href="#">nvmlViolationTime_t</a>	133



# Chapter 6

## Module Documentation

### 6.1 Device Structs

#### Data Structures

- struct [nvmlPciInfo\\_t](#)
- struct [nvmlEccErrorCounts\\_t](#)
- struct [nvmlUtilization\\_t](#)
- struct [nvmlMemory\\_t](#)
- struct [nvmlBAR1Memory\\_t](#)
- struct [nvmlProcessInfo\\_t](#)
- struct [nvmlNvLinkUtilizationControl\\_t](#)
- struct [nvmlBridgeChipInfo\\_t](#)
- struct [nvmlBridgeChipHierarchy\\_t](#)
- union [nvmlValue\\_t](#)
- struct [nvmlSample\\_t](#)
- struct [nvmlViolationTime\\_t](#)

#### Defines

- [#define NVML\\_VALUE\\_NOT\\_AVAILABLE \(-1\)](#)
- [#define NVML\\_DEVICE\\_PCI\\_BUS\\_ID\\_BUFFER\\_SIZE 16](#)
- [#define NVML\\_NVLINK\\_MAX\\_LINKS 4](#)
- [#define NVML\\_MAX\\_PHYSICAL\\_BRIDGE \(128\)](#)

#### Enumerations

- enum [nvmlBridgeChipType\\_t](#)
- enum [nvmlNvLinkUtilizationCountUnits\\_t](#)
- enum [nvmlNvLinkUtilizationCountPktTypes\\_t](#)
- enum [nvmlNvLinkCapability\\_t](#)
- enum [nvmlNvLinkErrorCounter\\_t](#)
- enum [nvmlGpuTopologyLevel\\_t](#)

- enum `nvmlSamplingType_t` {  
`NVML_TOTAL_POWER_SAMPLES` = 0,  
`NVML_GPU_UTILIZATION_SAMPLES` = 1,  
`NVML_MEMORY_UTILIZATION_SAMPLES` = 2,  
`NVML_ENC_UTILIZATION_SAMPLES` = 3,  
`NVML_DEC_UTILIZATION_SAMPLES` = 4,  
`NVML_PROCESSOR_CLK_SAMPLES` = 5,  
`NVML_MEMORY_CLK_SAMPLES` = 6 }
- enum `nvmlPcieUtilCounter_t`
- enum `nvmlValueType_t`
- enum `nvmlPerfPolicyType_t`

### 6.1.1 Define Documentation

#### 6.1.1.1 `#define NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE 16`

Buffer size guaranteed to be large enough for pci bus id

#### 6.1.1.2 `#define NVML_MAX_PHYSICAL_BRIDGE (128)`

Maximum limit on Physical Bridges per Board

#### 6.1.1.3 `#define NVML_NVLINK_MAX_LINKS 4`

Maximum number of NvLink links supported

#### 6.1.1.4 `#define NVML_VALUE_NOT_AVAILABLE (-1)`

Special constant that some fields take when they are not available. Used when only part of the struct is not available. Each structure explicitly states when to check for this value.

### 6.1.2 Enumeration Type Documentation

#### 6.1.2.1 enum `nvmlBridgeChipType_t`

Enum to represent type of bridge chip

#### 6.1.2.2 enum `nvmlGpuTopologyLevel_t`

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

#### 6.1.2.3 enum `nvmlNvLinkCapability_t`

Enum to represent NvLink queryable capabilities

**6.1.2.4 enum nvmlNvLinkErrorCounter\_t**

Enum to represent NvLink queryable error counters

**6.1.2.5 enum nvmlNvLinkUtilizationCountPktTypes\_t**

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

**6.1.2.6 enum nvmlNvLinkUtilizationCountUnits\_t**

Enum to represent the NvLink utilization counter packet units

**6.1.2.7 enum nvmlPcieUtilCounter\_t**

Represents the queryable PCIe utilization counters

**6.1.2.8 enum nvmlPerfPolicyType\_t**

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

**6.1.2.9 enum nvmlSamplingType\_t**

Represents Type of Sampling Event

**Enumerator:**

*NVML\_TOTAL\_POWER\_SAMPLES* To represent total power drawn by GPU.

*NVML\_GPU\_UTILIZATION\_SAMPLES* To represent percent of time during which one or more kernels was executing on the GPU.

*NVML\_MEMORY\_UTILIZATION\_SAMPLES* To represent percent of time during which global (device) memory was being read or written.

*NVML\_ENC\_UTILIZATION\_SAMPLES* To represent percent of time during which NVENC remains busy.

*NVML\_DEC\_UTILIZATION\_SAMPLES* To represent percent of time during which NVDEC remains busy.

*NVML\_PROCESSOR\_CLK\_SAMPLES* To represent processor clock samples.

*NVML\_MEMORY\_CLK\_SAMPLES* To represent memory clock samples.

**6.1.2.10 enum nvmlValueType\_t**

Represents the type for sample value returned

## 6.2 Device Enums

### Defines

- `#define nvmlFlagDefault 0x00`  
*Generic flag used to specify the default behavior of some functions. See description of particular functions for details.*
- `#define nvmlFlagForce 0x01`  
*Generic flag used to force some behavior. See description of particular functions for details.*
- `#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 nvmlBrandType_t`
- `enum nvmlTemperatureThresholds_t`
- `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,`  
    `NVML_ECC_COUNTER_TYPE_COUNT }`
- `enum nvmlClockType_t {`  
    `NVML_CLOCK_GRAPHICS = 0,`  
    `NVML_CLOCK_SM = 1,`  
    `NVML_CLOCK_MEM = 2,`  
    `NVML_CLOCK_VIDEO = 3 }`
- `enum nvmlClockId_t {`  
    `NVML_CLOCK_ID_CURRENT = 0,`  
    `NVML_CLOCK_ID_APP_CLOCK_TARGET = 1,`  
    `NVML_CLOCK_ID_APP_CLOCK_DEFAULT = 2,`  
    `NVML_CLOCK_ID_CUSTOMER_BOOST_MAX = 3 }`

- 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_GPU_IS_LOST = 15,
NVML_ERROR_RESET_REQUIRED = 16,
NVML_ERROR_OPERATING_SYSTEM = 17,
NVML_ERROR_LIB_RM_VERSION_MISMATCH = 18,
NVML_ERROR_IN_USE = 19,
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_TEXTURE_SHM = 5,
    NVML_MEMORY_LOCATION_COUNT }
• enum nvmlPageRetirementCause_t {
    NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS = 0,
    NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR = 1 }
• enum nvmlRestrictedAPI_t {
    NVML_RESTRICTED_API_SET_APPLICATION_CLOCKS = 0,
    NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS = 1 }

```

## 6.2.1 Define Documentation

### 6.2.1.1 #define NVML\_DOUBLE\_BIT\_ECC NVML\_MEMORY\_ERROR\_TYPE\_UNCORRECTED

Double bit ECC errors

#### Deprecated

Mapped to [NVML\\_MEMORY\\_ERROR\\_TYPE\\_UNCORRECTED](#)

### 6.2.1.2 #define NVML\_SINGLE\_BIT\_ECC NVML\_MEMORY\_ERROR\_TYPE\_CORRECTED

Single bit ECC errors

#### Deprecated

Mapped to [NVML\\_MEMORY\\_ERROR\\_TYPE\\_CORRECTED](#)



**6.2.1.3 #define nvmlEccBitType\_t nvmlMemoryErrorType\_t**

ECC bit types.

**Deprecated**

See [nvmlMemoryErrorType\\_t](#) for a more flexible type

**6.2.2 Enumeration Type Documentation****6.2.2.1 enum nvmlBrandType\_t**

\* The Brand of the GPU

**6.2.2.2 enum nvmlClockId\_t**

Clock Ids. These are used in combination with nvmlClockType\_t to specify a single clock value.

**Enumerator:**

*NVML\_CLOCK\_ID\_CURRENT* Current actual clock value.  
*NVML\_CLOCK\_ID\_APP\_CLOCK\_TARGET* Target application clock.  
*NVML\_CLOCK\_ID\_APP\_CLOCK\_DEFAULT* Default application clock target.  
*NVML\_CLOCK\_ID\_CUSTOMER\_BOOST\_MAX* OEM-defined maximum clock rate.

**6.2.2.3 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.  
*NVML\_CLOCK\_VIDEO* Video encoder/decoder clock domain.

**6.2.2.4 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* Support Removed.  
*NVML\_COMPUTEMODE\_PROHIBITED* Compute-prohibited mode – no contexts per device.  
*NVML\_COMPUTEMODE\_EXCLUSIVE\_PROCESS* Compute-exclusive-process mode – only one context per device, usable from multiple threads at a time.

### 6.2.2.5 enum nvmlDriverModel\_t

Driver models.

Windows only.

#### Enumerator:

**NVML\_DRIVER\_WDDM** WDDM driver model – GPU treated as a display device.

**NVML\_DRIVER\_WDM** WDM (TCC) model (recommended) – GPU treated as a generic device.

### 6.2.2.6 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).

**NVML\_ECC\_COUNTER\_TYPE\_COUNT** Count of memory counter types.

### 6.2.2.7 enum nvmlEnableState\_t

Generic enable/disable enum.

#### Enumerator:

**NVML\_FEATURE\_DISABLED** Feature disabled.

**NVML\_FEATURE\_ENABLED** Feature enabled.

### 6.2.2.8 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.

**6.2.2.9 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.

**6.2.2.10 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.

**6.2.2.11 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\_TEXTURE\_SHM* Shared memory.
- NVML\_MEMORY\_LOCATION\_COUNT* This counts the number of memory locations the driver knows about.

**6.2.2.12 enum nvmlPageRetirementCause\_t**

Causes for page retirement

**Enumerator:**

- NVML\_PAGE\_RETIREMENT\_CAUSE\_MULTIPLE\_SINGLE\_BIT\_ECC\_ERRORS* Page was retired due to multiple single bit ECC error.
- NVML\_PAGE\_RETIREMENT\_CAUSE\_DOUBLE\_BIT\_ECC\_ERROR* Page was retired due to double bit ECC error.

### 6.2.2.13 enum nvmlPstates\_t

Allowed PStates.

#### Enumerator:

**NVML\_PSTATE\_0** Performance state 0 – Maximum Performance.  
**NVML\_PSTATE\_1** Performance state 1.  
**NVML\_PSTATE\_2** Performance state 2.  
**NVML\_PSTATE\_3** Performance state 3.  
**NVML\_PSTATE\_4** Performance state 4.  
**NVML\_PSTATE\_5** Performance state 5.  
**NVML\_PSTATE\_6** Performance state 6.  
**NVML\_PSTATE\_7** Performance state 7.  
**NVML\_PSTATE\_8** Performance state 8.  
**NVML\_PSTATE\_9** Performance state 9.  
**NVML\_PSTATE\_10** Performance state 10.  
**NVML\_PSTATE\_11** Performance state 11.  
**NVML\_PSTATE\_12** Performance state 12.  
**NVML\_PSTATE\_13** Performance state 13.  
**NVML\_PSTATE\_14** Performance state 14.  
**NVML\_PSTATE\_15** Performance state 15 – Minimum Performance.  
**NVML\_PSTATE\_UNKNOWN** Unknown performance state.

### 6.2.2.14 enum nvmlRestrictedAPI\_t

API types that allow changes to default permission restrictions

#### Enumerator:

**NVML\_RESTRICTED\_API\_SET\_APPLICATION\_CLOCKS** APIs that change application clocks, see `nvmlDeviceSetApplicationsClocks <` and see `nvmlDeviceResetApplicationsClocks`.  
**NVML\_RESTRICTED\_API\_SET\_AUTO\_BOOSTED\_CLOCKS** APIs that enable/disable Auto Boosted clocks < see `nvmlDeviceSetAutoBoostedClocksEnabled`.

### 6.2.2.15 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\_GPU\_IS\_LOST*** The GPU has fallen off the bus or has otherwise become inaccessible.

***NVML\_ERROR\_RESET\_REQUIRED*** The GPU requires a reset before it can be used again.

***NVML\_ERROR\_OPERATING\_SYSTEM*** The GPU control device has been blocked by the operating system/cgroups.

***NVML\_ERROR\_LIB\_RM\_VERSION\_MISMATCH*** RM detects a driver/library version mismatch.

***NVML\_ERROR\_IN\_USE*** An operation cannot be performed because the GPU is currently in use.

***NVML\_ERROR\_UNKNOWN*** An internal driver error occurred.

#### 6.2.2.16 enum nvmlTemperatureSensors\_t

Temperature sensors.

##### Enumerator:

***NVML\_TEMPERATURE\_GPU*** Temperature sensor for the GPU die.

#### 6.2.2.17 enum nvmlTemperatureThresholds\_t

Temperature thresholds.

## 6.3 Unit Structs

### Data Structures

- struct [nvmlHwbcEntry\\_t](#)
- struct [nvmlLedState\\_t](#)
- struct [nvmlUnitInfo\\_t](#)
- struct [nvmlPSUInfo\\_t](#)
- struct [nvmlUnitFanInfo\\_t](#)
- struct [nvmlUnitFanSpeeds\\_t](#)

### Enumerations

- enum [nvmlFanState\\_t](#) {  
    [NVML\\_FAN\\_NORMAL](#) = 0,  
    [NVML\\_FAN\\_FAILED](#) = 1 }
- enum [nvmlLedColor\\_t](#) {  
    [NVML\\_LED\\_COLOR\\_GREEN](#) = 0,  
    [NVML\\_LED\\_COLOR\\_AMBER](#) = 1 }

#### 6.3.1 Enumeration Type Documentation

##### 6.3.1.1 enum [nvmlFanState\\_t](#)

Fan state enum.

###### Enumerator:

***NVML\_FAN\_NORMAL*** Fan is working properly.

***NVML\_FAN\_FAILED*** Fan has failed.

##### 6.3.1.2 enum [nvmlLedColor\\_t](#)

Led color enum.

###### Enumerator:

***NVML\_LED\_COLOR\_GREEN*** GREEN, indicates good health.

***NVML\_LED\_COLOR\_AMBER*** AMBER, indicates problem.

## 6.4 Event Types

### Defines

- `#define nvmlEventTypeSingleBitEccError 0x0000000000000001LL`  
*Event about single bit ECC errors.*
- `#define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL`  
*Event about double bit ECC errors.*
- `#define nvmlEventTypePState 0x0000000000000004LL`  
*Event about PState changes.*
- `#define nvmlEventTypeXidCriticalError 0x0000000000000008LL`  
*Event that Xid critical error occurred.*
- `#define nvmlEventTypeClock 0x0000000000000010LL`  
*Event about clock changes.*
- `#define nvmlEventTypeNone 0x0000000000000000LL`  
*Mask with no events.*
- `#define nvmlEventTypeAll`  
*Mask of all events.*

### 6.4.1 Detailed Description

Event Types which user can be notified about. See description of particular functions for details.

See [nvmlDeviceRegisterEvents](#) and [nvmlDeviceGetSupportedEventTypes](#) to check which devices support each event.

Types can be combined with bitwise or operator `'|'` when passed to [nvmlDeviceRegisterEvents](#)

### 6.4.2 Define Documentation

#### 6.4.2.1 `#define nvmlEventTypeClock 0x0000000000000010LL`

Kepler only

#### 6.4.2.2 `#define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL`

#### Note:

An uncorrected texture memory error is not an ECC error, so it does not generate a double bit event

**6.4.2.3 #define nvmlEventTypePState 0x0000000000000004LL****Note:**

On Fermi architecture PState changes are also an indicator that GPU is throttling down due to no work being executed on the GPU, power capping or thermal capping. In a typical situation, Fermi-based GPU should stay in P0 for the duration of the execution of the compute process.

**6.4.2.4 #define nvmlEventTypeSingleBitEccError 0x0000000000000001LL****Note:**

A corrected texture memory error is not an ECC error, so it does not generate a single bit event



## 6.5 Accounting Statistics

### Data Structures

- struct [nvmlAccountingStats\\_t](#)

### Functions

- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceGetAccountingMode](#) (nvmlDevice\_t device, [nvmlEnableState\\_t](#) \*mode)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceGetAccountingStats](#) (nvmlDevice\_t device, unsigned int pid, [nvmlAccountingStats\\_t](#) \*stats)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceGetAccountingPids](#) (nvmlDevice\_t device, unsigned int \*count, unsigned int \*pids)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceGetAccountingBufferSize](#) (nvmlDevice\_t device, unsigned int \*bufferSize)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceSetAccountingMode](#) (nvmlDevice\_t device, [nvmlEnableState\\_t](#) mode)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceClearAccountingPids](#) (nvmlDevice\_t device)

#### 6.5.1 Detailed Description

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

#### Note:

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

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

#### 6.5.2 Function Documentation

##### 6.5.2.1 [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceClearAccountingPids](#) (nvmlDevice\_t device)

Clears accounting information about all processes that have already terminated.

For Kepler <sup>TM</sup> or newer fully supported devices. Requires root/admin permissions.

See [nvmlDeviceGetAccountingMode](#) See [nvmlDeviceGetAccountingStats](#) See [nvmlDeviceSetAccountingMode](#)

#### Parameters:

*device* The identifier of the target device

#### Returns:

- [NVML\\_SUCCESS](#) if accounting information has been cleared
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* are invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.5.2.2 `nvmlReturn_t DECLDIR nvmlDeviceGetAccountingBufferSize (nvmlDevice_t device, unsigned int * bufferSize)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

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

#### Parameters:

*device* The identifier of the target device

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

#### Returns:

- [NVML\\_SUCCESS](#) if buffer size was successfully retrieved
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *bufferSize* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature or accounting mode is disabled
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlDeviceGetAccountingStats](#)

[nvmlDeviceGetAccountingPids](#)

### 6.5.2.3 `nvmlReturn_t DECLDIR nvmlDeviceGetAccountingMode (nvmlDevice_t device, nvmlEnableState_t * mode)`

Queries the state of per process accounting mode.

For Kepler <sup>TM</sup> or newer fully supported devices.

See [nvmlDeviceGetAccountingStats](#) for more details. See [nvmlDeviceSetAccountingMode](#)

#### Parameters:

*device* The identifier of the target device

*mode* Reference in which to return the current accounting mode

#### Returns:

- [NVML\\_SUCCESS](#) if the mode has been successfully retrieved
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *mode* are NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.5.2.4 `nvmlReturn_t DECLDIR nvmlDeviceGetAccountingPids (nvmlDevice_t device, unsigned int * count, unsigned int * pids)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

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

For more details see [nvmlDeviceGetAccountingStats](#).

#### Note:

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

#### Parameters:

**device** The identifier of the target device

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

**pids** Reference in which to return list of process ids

#### Returns:

- [NVML\\_SUCCESS](#) if pids were successfully retrieved
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *count* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature or accounting mode is disabled
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *count* is too small (*count* is set to expected value)
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlDeviceGetAccountingBufferSize](#)

#### 6.5.2.5 `nvmlReturn_t DECLDIR nvmlDeviceGetAccountingStats (nvmlDevice_t device, unsigned int pid, nvmlAccountingStats_t * stats)`

Queries process's accounting stats.

For Kepler <sup>TM</sup> or newer fully supported devices.

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

See [nvmlAccountingStats\\_t](#) for description of each returned metric. List of processes that can be queried can be retrieved from [nvmlDeviceGetAccountingPids](#).

#### Note:

Accounting Mode needs to be on. See [nvmlDeviceGetAccountingMode](#).

Only compute and graphics applications stats can be queried. Monitoring applications stats can't be queried since they don't contribute to GPU utilization.

In case of pid collision stats of only the latest process (that terminated last) will be reported

**Warning:**

On Kepler devices per process statistics are accurate only if there's one process running on a GPU.

**Parameters:**

*device* The identifier of the target device  
*pid* Process Id of the target process to query stats for  
*stats* Reference in which to return the process's accounting stats

**Returns:**

- [NVML\\_SUCCESS](#) if stats have been successfully retrieved
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *stats* are NULL
- [NVML\\_ERROR\\_NOT\\_FOUND](#) if process stats were not found
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature or accounting mode is disabled
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[nvmlDeviceGetAccountingBufferSize](#)

#### 6.5.2.6 `nvmlReturn_t DECLDIR nvmlDeviceSetAccountingMode (nvmlDevice_t device, nvmlEnableState_t mode)`

Enables or disables per process accounting.

For Kepler <sup>TM</sup> or newer fully supported devices. Requires root/admin permissions.

**Note:**

This setting is not persistent and will default to disabled after driver unloads. Enable persistence mode to be sure the setting doesn't switch off to disabled.

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

Disabling accounting clears all accounting pids information.

See [nvmlDeviceGetAccountingMode](#) See [nvmlDeviceGetAccountingStats](#) See [nvmlDeviceClearAccountingPids](#)

**Parameters:**

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

**Returns:**

- [NVML\\_SUCCESS](#) if the new mode has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* or *mode* are invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

## 6.6 Initialization and Cleanup

### Functions

- [nvmlReturn\\_t](#) DECLDIR [nvmlInit](#) (void)
- [nvmlReturn\\_t](#) DECLDIR [nvmlShutdown](#) (void)

#### 6.6.1 Detailed Description

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

#### 6.6.2 Function Documentation

##### 6.6.2.1 [nvmlReturn\\_t](#) DECLDIR [nvmlInit](#) (void)

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

**Note:**

In NVML 5.319 new [nvmlInit\\_v2](#) has replaced [nvmlInit\\_v1](#) (default in NVML 4.304 and older) that did initialize all GPU devices in the system.

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

**Note:**

To contrast [nvmlInit\\_v2](#) with [nvmlInit\\_v1](#), NVML 4.304 [nvmlInit\\_v1](#) will fail when any detected GPU is in a bad or unstable state.

For all products.

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

**Returns:**

- [NVML\\_SUCCESS](#) if NVML has been properly initialized
- [NVML\\_ERROR\\_DRIVER\\_NOT\\_LOADED](#) if NVIDIA driver is not running
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if NVML does not have permission to talk to the driver
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

##### 6.6.2.2 [nvmlReturn\\_t](#) DECLDIR [nvmlShutdown](#) (void)

Shut down NVML by releasing all GPU resources previously allocated with [nvmlInit\(\)](#).

For all products.

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

**Returns:**

- [NVML\\_SUCCESS](#) if NVML has been properly shut down
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

## 6.7 Error reporting

### Functions

- `const DECLDIR char * nvmlErrorString (nvmlReturn\_t result)`

#### 6.7.1 Detailed Description

This chapter describes helper functions for error reporting routines.

#### 6.7.2 Function Documentation

##### 6.7.2.1 `const DECLDIR char* nvmlErrorString (nvmlReturn_t result)`

Helper method for converting NVML error codes into readable strings.

For all products.

##### Parameters:

*result* NVML error code to convert

##### Returns:

String representation of the error.

## 6.8 Constants

### Defines

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

### 6.8.1 Define Documentation

#### 6.8.1.1 `#define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16`

Buffer size guaranteed to be large enough for [nvmlDeviceGetInforomVersion](#) and [nvmlDeviceGetInforomImageVersion](#)

#### 6.8.1.2 `#define NVML_DEVICE_NAME_BUFFER_SIZE 64`

Buffer size guaranteed to be large enough for [nvmlDeviceGetName](#)

#### 6.8.1.3 `#define NVML_DEVICE_PART_NUMBER_BUFFER_SIZE 80`

Buffer size guaranteed to be large enough for [nvmlDeviceGetBoardPartNumber](#)

#### 6.8.1.4 `#define NVML_DEVICE_SERIAL_BUFFER_SIZE 30`

Buffer size guaranteed to be large enough for [nvmlDeviceGetSerial](#)

#### 6.8.1.5 `#define NVML_DEVICE_UUID_BUFFER_SIZE 80`

Buffer size guaranteed to be large enough for [nvmlDeviceGetUUID](#)

#### 6.8.1.6 `#define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32`

Buffer size guaranteed to be large enough for [nvmlDeviceGetVbiosVersion](#)

#### 6.8.1.7 `#define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80`

Buffer size guaranteed to be large enough for [nvmlSystemGetDriverVersion](#)

#### 6.8.1.8 `#define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80`

Buffer size guaranteed to be large enough for [nvmlSystemGetNVMLVersion](#)



## 6.9 System Queries

### Functions

- [nvmlReturn\\_t](#) DECLDIR [nvmlSystemGetDriverVersion](#) (char \*version, unsigned int length)
- [nvmlReturn\\_t](#) DECLDIR [nvmlSystemGetNVMLVersion](#) (char \*version, unsigned int length)
- [nvmlReturn\\_t](#) DECLDIR [nvmlSystemGetProcessName](#) (unsigned int pid, char \*name, unsigned int length)

### 6.9.1 Detailed Description

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

### 6.9.2 Function Documentation

#### 6.9.2.1 [nvmlReturn\\_t](#) DECLDIR [nvmlSystemGetDriverVersion](#) (char \* *version*, unsigned int *length*)

Retrieves the version of the system's graphics driver.

For all products.

The version identifier is an alphanumeric string. It will not exceed 80 characters in length (including the NULL terminator). See [nvmlConstants::NVML\\_SYSTEM\\_DRIVER\\_VERSION\\_BUFFER\\_SIZE](#).

#### Parameters:

*version* Reference in which to return the version identifier

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

#### Returns:

- [NVML\\_SUCCESS](#) if *version* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *version* is NULL
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *length* is too small

#### 6.9.2.2 [nvmlReturn\\_t](#) DECLDIR [nvmlSystemGetNVMLVersion](#) (char \* *version*, unsigned int *length*)

Retrieves the version of the NVML library.

For all products.

The version identifier is an alphanumeric string. It will not exceed 80 characters in length (including the NULL terminator). See [nvmlConstants::NVML\\_SYSTEM\\_NVML\\_VERSION\\_BUFFER\\_SIZE](#).

#### Parameters:

*version* Reference in which to return the version identifier

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

#### Returns:

- [NVML\\_SUCCESS](#) if *version* has been set

- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *version* is NULL
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *length* is too small

### 6.9.2.3 `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 or *length* is 0.
- [NVML\\_ERROR\\_NOT\\_FOUND](#) if process doesn't exist
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

## 6.10 Unit Queries

### Functions

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

### 6.10.1 Detailed Description

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

### 6.10.2 Function Documentation

#### 6.10.2.1 `nvmlReturn_t` DECLDIR `nvmlSystemGetHicVersion` (unsigned int \* *hwbcCount*, `nvmlHwbcEntry_t` \* *hwbcEntries*)

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

For S-class products.

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

#### Parameters:

*hwbcCount* Size of *hwbcEntries* array

*hwbcEntries* Array holding information about hwbc

#### Returns:

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

#### 6.10.2.2 `nvmlReturn_t` DECLDIR `nvmlUnitGetCount` (unsigned int \* *unitCount*)

Retrieves the number of units in the system.

For S-class products.

**Parameters:**

*unitCount* Reference in which to return the number of units

**Returns:**

- [NVML\\_SUCCESS](#) if *unitCount* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *unitCount* is NULL
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.10.2.3 `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
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

Retrieves the fan speed readings for the unit.

For S-class products.

See [nvmlUnitFanSpeeds\\_t](#) for details on available fan speed info.

**Parameters:**

*unit* The identifier of the target unit

*fanSpeeds* Reference in which to return the fan speed information

**Returns:**

- [NVML\\_SUCCESS](#) if *fanSpeeds* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *unit* is invalid or *fanSpeeds* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this is not an S-class product
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**6.10.2.5 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,  $\geq 0$  and  $< unitCount$

*unit* Reference in which to return the unit handle

**Returns:**

- [NVML\\_SUCCESS](#) if *unit* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *index* is invalid or *unit* is NULL
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**6.10.2.6 nvmlReturn\_t DECLDIR nvmlUnitGetLedState (nvmlUnit\_t *unit*, nvmlLedState\_t \* *state*)**

Retrieves the LED state associated with this unit.

For S-class products.

See [nvmlLedState\\_t](#) for details on allowed states.

**Parameters:**

*unit* The identifier of the target unit

*state* Reference in which to return the current LED state

**Returns:**

- [NVML\\_SUCCESS](#) if *state* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *unit* is invalid or *state* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this is not an S-class product
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[nvmlUnitSetLedState\(\)](#)

**6.10.2.7 nvmlReturn\_t DECLDIR nvmlUnitGetPsuInfo (nvmlUnit\_t *unit*, nvmlPSUInfo\_t \* *psu*)**

Retrieves the PSU stats for the unit.

For S-class products.

See [nvmlPSUInfo\\_t](#) for details on available PSU info.

**Parameters:**

- unit* The identifier of the target unit
- psu* Reference in which to return the PSU information

**Returns:**

- [NVML\\_SUCCESS](#) if *psu* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *unit* is invalid or *psu* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this is not an S-class product
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

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

For S-class products.

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

**Parameters:**

- unit* The identifier of the target unit
- type* The type of reading to take
- temp* Reference in which to return the intake temperature

**Returns:**

- [NVML\\_SUCCESS](#) if *temp* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *unit* or *type* is invalid or *temp* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this is not an S-class product
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

Retrieves the static information associated with a unit.

For S-class products.

See [nvmlUnitInfo\\_t](#) for details on available unit info.

**Parameters:**

- unit* The identifier of the target unit
- info* Reference in which to return the unit information

**Returns:**

- [NVML\\_SUCCESS](#) if *info* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *unit* is invalid or *info* is NULL

## 6.11 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 `nvmlDeviceGetBrand` (`nvmlDevice_t` device, `nvmlBrandType_t` \*type)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetIndex` (`nvmlDevice_t` device, unsigned int \*index)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetSerial` (`nvmlDevice_t` device, char \*serial, unsigned int length)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetCpuAffinity` (`nvmlDevice_t` device, unsigned int cpuSetSize, unsigned long \*cpuSet)
- `nvmlReturn_t` DECLDIR `nvmlDeviceSetCpuAffinity` (`nvmlDevice_t` device)
- `nvmlReturn_t` DECLDIR `nvmlDeviceClearCpuAffinity` (`nvmlDevice_t` device)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetTopologyCommonAncestor` (`nvmlDevice_t` device1, `nvmlDevice_t` device2, `nvmlGpuTopologyLevel_t` \*pathInfo)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetTopologyNearestGpus` (`nvmlDevice_t` device, `nvmlGpuTopologyLevel_t` level, unsigned int \*count, `nvmlDevice_t` \*deviceArray)
- `nvmlReturn_t` DECLDIR `nvmlSystemGetTopologyGpuSet` (unsigned int cpuNumber, unsigned int \*count, `nvmlDevice_t` \*deviceArray)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetUUID` (`nvmlDevice_t` device, char \*uuid, unsigned int length)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetMinorNumber` (`nvmlDevice_t` device, unsigned int \*minorNumber)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetBoardPartNumber` (`nvmlDevice_t` device, char \*partNumber, 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 `nvmlDeviceGetDisplayActive` (`nvmlDevice_t` device, `nvmlEnableState_t` \*isActive)
- `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 `nvmlDeviceGetPcieThroughput` (`nvmlDevice_t` device, `nvmlPcieUtilCounter_t` counter, unsigned int \*value)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPcieReplayCounter` (`nvmlDevice_t` device, unsigned int \*value)
- `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 [nvmlDeviceGetDefaultApplicationsClock](#) (nvmlDevice\_t device, [nvmlClockType\\_t](#) clockType, unsigned int \*clockMHz)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceResetApplicationsClocks](#) (nvmlDevice\_t device)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceGetClock](#) (nvmlDevice\_t device, [nvmlClockType\\_t](#) clockType, [nvmlClockId\\_t](#) clockId, unsigned int \*clockMHz)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceGetMaxCustomerBoostClock](#) (nvmlDevice\_t device, [nvmlClockType\\_t](#) clockType, unsigned int \*clockMHz)
- [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 [nvmlDeviceGetAutoBoostedClocksEnabled](#) (nvmlDevice\_t device, [nvmlEnableState\\_t](#) \*isEnabled, [nvmlEnableState\\_t](#) \*defaultIsEnabled)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceSetAutoBoostedClocksEnabled](#) (nvmlDevice\_t device, [nvmlEnableState\\_t](#) enabled)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceSetDefaultAutoBoostedClocksEnabled](#) (nvmlDevice\_t device, [nvmlEnableState\\_t](#) enabled, unsigned int flags)
- [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 [nvmlDeviceGetTemperatureThreshold](#) (nvmlDevice\_t device, [nvmlTemperatureThresholds\\_t](#) thresholdType, 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 [nvmlDeviceGetEnforcedPowerLimit](#) (nvmlDevice\_t device, unsigned int \*limit)
- [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 [nvmlDeviceGetBoardId](#) (nvmlDevice\_t device, unsigned int \*boardId)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceGetMultiGpuBoard](#) (nvmlDevice\_t device, unsigned int \*multiGpuBool)
- [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 nvmlDeviceGetEncoderUtilization` (`nvmlDevice_t` device, unsigned int \*utilization, unsigned int \*samplingPeriodUs)
- `nvmlReturn_t DECLDIR nvmlDeviceGetDecoderUtilization` (`nvmlDevice_t` device, unsigned int \*utilization, unsigned int \*samplingPeriodUs)
- `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 nvmlDeviceGetBridgeChipInfo` (`nvmlDevice_t` device, `nvmlBridgeChipHierarchy_t` \*bridgeHierarchy)
- `nvmlReturn_t DECLDIR nvmlDeviceGetComputeRunningProcesses` (`nvmlDevice_t` device, unsigned int \*infoCount, `nvmlProcessInfo_t` \*infos)
- `nvmlReturn_t DECLDIR nvmlDeviceGetGraphicsRunningProcesses` (`nvmlDevice_t` device, unsigned int \*infoCount, `nvmlProcessInfo_t` \*infos)
- `nvmlReturn_t DECLDIR nvmlDeviceOnSameBoard` (`nvmlDevice_t` device1, `nvmlDevice_t` device2, int \*onSameBoard)
- `nvmlReturn_t DECLDIR nvmlDeviceGetAPIRestriction` (`nvmlDevice_t` device, `nvmlRestrictedAPI_t` apiType, `nvmlEnableState_t` \*isRestricted)
- `nvmlReturn_t DECLDIR nvmlDeviceGetSamples` (`nvmlDevice_t` device, `nvmlSamplingType_t` type, unsigned long long lastSeenTimeStamp, `nvmlValueType_t` \*sampleValType, unsigned int \*sampleCount, `nvmlSample_t` \*samples)
- `nvmlReturn_t DECLDIR nvmlDeviceGetBAR1MemoryInfo` (`nvmlDevice_t` device, `nvmlBAR1Memory_t` \*bar1Memory)
- `nvmlReturn_t DECLDIR nvmlDeviceGetViolationStatus` (`nvmlDevice_t` device, `nvmlPerfPolicyType_t` perfPolicyType, `nvmlViolationTime_t` \*violTime)
- `nvmlReturn_t DECLDIR nvmlDeviceGetRetiredPages` (`nvmlDevice_t` device, `nvmlPageRetirementCause_t` cause, unsigned int \*pageCount, unsigned long long \*addresses)
- `nvmlReturn_t DECLDIR nvmlDeviceGetRetiredPagesPendingStatus` (`nvmlDevice_t` device, `nvmlEnableState_t` \*isPending)

### 6.11.1 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()`, `nvmlDeviceGetHandleByPciBusId()`. or `nvmlDeviceGetHandleByUUID()`.

### 6.11.2 Function Documentation

#### 6.11.2.1 `nvmlReturn_t DECLDIR nvmlDeviceClearCpuAffinity` (`nvmlDevice_t` device)

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

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

**Parameters:**

*device* The identifier of the target device

**Returns:**

- [NVML\\_SUCCESS](#) if the calling process has been successfully unbound
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.2 `nvmlReturn_t DECLDIR nvmlDeviceGetAPIRestriction (nvmlDevice_t device, nvmlRestrictedAPI_t apiType, nvmlEnableState_t * isRestricted)`

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

For all fully supported products.

**Parameters:**

*device* The identifier of the target device

*apiType* Target API type for this operation

*isRestricted* Reference in which to return the current restriction NVML\_FEATURE\_ENABLED indicates that the API is root-only NVML\_FEATURE\_DISABLED indicates that the API is accessible to all users

**Returns:**

- [NVML\\_SUCCESS](#) if *isRestricted* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, *apiType* incorrect or *isRestricted* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this query is not supported by the device or the device does not support the feature that is being queried (E.G. Enabling/disabling Auto Boosted clocks is not supported by the device)
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[nvmlRestrictedAPI\\_t](#)

### 6.11.2.3 `nvmlReturn_t DECLDIR nvmlDeviceGetApplicationsClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int * clockMHz)`

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

For Kepler™ or newer fully supported devices.

**Parameters:**

*device* The identifier of the target device

*clockType* Identify which clock domain to query

*clockMHz* Reference in which to return the clock in MHz

#### Returns:

- [NVML\\_SUCCESS](#) if *clockMHz* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *clockMHz* is NULL or *clockType* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.4 `nvmlReturn_t DECLDIR nvmlDeviceGetAutoBoostedClocksEnabled (nvmlDevice_t device, nvmlEnableState_t * isEnabled, nvmlEnableState_t * defaultIsEnabled)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

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

On Pascal and newer hardware, Auto Boosted clocks are controlled through application clocks. Use [nvmlDeviceSetApplicationsClocks](#) and [nvmlDeviceResetApplicationsClocks](#) to control Auto Boost behavior.

#### Parameters:

*device* The identifier of the target device

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

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

#### Returns:

- [NVML\\_SUCCESS](#) If *isEnabled* has been set with the Auto Boosted clocks state of *device*
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *isEnabled* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support Auto Boosted clocks
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.5 `nvmlReturn_t DECLDIR nvmlDeviceGetBAR1MemoryInfo (nvmlDevice_t device, nvmlBAR1Memory_t * bar1Memory)`

Gets Total, Available and Used size of BAR1 memory.

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

For Kepler <sup>TM</sup> or newer fully supported devices.

**Parameters:**

*device* The identifier of the target device

*bar1Memory* Reference in which BAR1 memory information is returned.

**Returns:**

- [NVML\\_SUCCESS](#) if BAR1 memory is successfully retrieved
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, *bar1Memory* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this query is not supported by the device
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.6 `nvmlReturn_t DECLDIR nvmlDeviceGetBoardId (nvmlDevice_t device, unsigned int * boardId)`

Retrieves the device boardId from 0-N. Devices with the same boardId indicate GPUs connected to the same PLX. Use in conjunction with [nvmlDeviceGetMultiGpuBoard\(\)](#) to decide if they are on the same board as well. The boardId returned is a unique ID for the current configuration. Uniqueness and ordering across reboots and system configurations is not guaranteed (i.e. if a Tesla K40c returns 0x100 and the two GPUs on a Tesla K10 in the same system returns 0x200 it is not guaranteed they will always return those values but they will always be different from each other).

For Fermi <sup>TM</sup> or newer fully supported devices.

**Parameters:**

*device* The identifier of the target device

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

**Returns:**

- [NVML\\_SUCCESS](#) if *boardId* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *boardId* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.7 `nvmlReturn_t DECLDIR nvmlDeviceGetBoardPartNumber (nvmlDevice_t device, char * partNumber, unsigned int length)`

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

For all products.

**Parameters:**

*device* Identifier of the target device

*partNumber* Reference to the buffer to return

*length* Length of the buffer reference

**Returns:**

- [NVML\\_SUCCESS](#) if *partNumber* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the needed VBIOS fields have not been filled
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *serial* is NULL
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**6.11.2.8 nvmlReturn\_t DECLDIR nvmlDeviceGetBrand (nvmlDevice\_t device, nvmlBrandType\_t \* type)**

Retrieves the brand of this device.

For all products.

The type is a member of [nvmlBrandType\\_t](#) defined above.

**Parameters:**

- device* The identifier of the target device
- type* Reference in which to return the product brand type

**Returns:**

- [NVML\\_SUCCESS](#) if *name* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, or *type* is NULL
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**6.11.2.9 nvmlReturn\_t DECLDIR nvmlDeviceGetBridgeChipInfo (nvmlDevice\_t device, nvmlBridgeChipHierarchy\_t \* bridgeHierarchy)**

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

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

**Parameters:**

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

**Returns:**

- [NVML\\_SUCCESS](#) if bridge chip exists
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, or *bridgeInfo* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if bridge chip not supported on the device
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.10 `nvmlReturn_t DECLDIR nvmlDeviceGetClock (nvmlDevice_t device, nvmlClockType_t clockType, nvmlClockId_t clockId, unsigned int * clockMHz)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

##### Parameters:

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

##### Returns:

- [NVML\\_SUCCESS](#) if *clockMHz* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *clockMHz* is NULL or *clockType* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.11 `nvmlReturn_t DECLDIR nvmlDeviceGetClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int * clock)`

Retrieves the current clock speeds for the device.

For Fermi <sup>TM</sup> or newer fully supported devices.

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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.12 `nvmlReturn_t DECLDIR nvmlDeviceGetComputeMode (nvmlDevice_t device, nvmlComputeMode_t * mode)`

Retrieves the current compute mode for the device.

For all 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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlDeviceSetComputeMode\(\)](#)

### 6.11.2.13 `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 Fermi <sup>TM</sup> or newer fully supported devices.

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

To query the current number of running compute processes, call this function with *\*infoCount* = 0. The return code will be [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#), or [NVML\\_SUCCESS](#) if none are running. For this call *infos* is allowed to be NULL.

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

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

#### Parameters:

*device* The identifier of the target device

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

*infos* Reference in which to return the process information

#### Returns:

- [NVML\\_SUCCESS](#) if *infoCount* and *infos* have been populated

- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *infoCount* indicates that the *infos* array is too small *infoCount* will contain minimal amount of space necessary for the call to complete
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, either of *infoCount* or *infos* is NULL
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlSystemGetProcessName](#)

#### 6.11.2.14 `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.

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

**Parameters:**

*deviceCount* Reference in which to return the number of accessible devices

**Returns:**

- [NVML\\_SUCCESS](#) if *deviceCount* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *deviceCount* is NULL
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.15 `nvmlReturn_t DECLDIR nvmlDeviceGetCpuAffinity (nvmlDevice_t device, unsigned int cpuSetSize, unsigned long * cpuSet)`

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

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

**Parameters:**

*device* The identifier of the target device

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

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

**Returns:**

- [NVML\\_SUCCESS](#) if *cpuAffinity* has been filled



- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, `cpuSetSize == 0`, or `cpuSet` is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.16 `nvmlReturn_t DECLDIR nvmlDeviceGetCurrentClocksThrottleReasons (nvmlDevice_t device, unsigned long long * clocksThrottleReasons)`

Retrieves current clocks throttling reasons.

For all fully supported products.

##### 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 set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *clocksThrottleReasons* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

##### See also:

[NvmlClocksThrottleReasons](#)

[nvmlDeviceGetSupportedClocksThrottleReasons](#)

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

Retrieves the current PCIe link generation

For Fermi <sup>TM</sup> or newer fully supported devices.

##### Parameters:

*device* The identifier of the target device

*currLinkGen* Reference in which to return the current PCIe link generation

##### Returns:

- [NVML\\_SUCCESS](#) if *currLinkGen* has been populated

- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *currLinkGen* is null
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if PCIe link information is not available
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

Retrieves the current PCIe link width

For Fermi <sup>TM</sup> or newer fully supported devices.

##### Parameters:

*device* The identifier of the target device

*currLinkWidth* Reference in which to return the current PCIe link generation

##### Returns:

- [NVML\\_SUCCESS](#) if *currLinkWidth* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *currLinkWidth* is null
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if PCIe link information is not available
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.19 `nvmlReturn_t DECLDIR nvmlDeviceGetDecoderUtilization (nvmlDevice_t device, unsigned int * utilization, unsigned int * samplingPeriodUs)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

##### Parameters:

*device* The identifier of the target device

*utilization* Reference to an unsigned int for decoder utilization info

*samplingPeriodUs* Reference to an unsigned int for the sampling period in US

##### Returns:

- [NVML\\_SUCCESS](#) if *utilization* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, *utilization* is NULL, or *samplingPeriodUs* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.20 `nvmlReturn_t DECLDIR nvmlDeviceGetDefaultApplicationsClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int * clockMHz)`

Retrieves the default applications clock that GPU boots with or defaults to after [nvmlDeviceResetApplicationsClocks](#) call.

For Kepler <sup>TM</sup> or newer fully supported devices.

#### Parameters:

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

#### Returns:

- [NVML\\_SUCCESS](#) if *clockMHz* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *clockMHz* is NULL or *clockType* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlDeviceGetApplicationsClock](#)

### 6.11.2.21 `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 Fermi <sup>TM</sup> or newer fully supported devices. Only applicable to devices with ECC. Requires [NVML\\_INFOROM\\_ECC](#) version 2.0 or higher to report aggregate location-based ECC counts. Requires [NVML\\_INFOROM\\_ECC](#) version 1.0 or higher to report all other ECC counts. Requires ECC Mode to be enabled.

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

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

See [nvmlMemoryErrorType\\_t](#) for a description of available bit types.

See [nvmlEccCounterType\\_t](#) for a description of available counter types.

See [nvmlEccErrorCounts\\_t](#) for a description of provided detailed ECC counts.

#### Parameters:

- device* The identifier of the target device

***errorType*** Flag that specifies the type of the errors.

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

***eccCounts*** Reference in which to return the specified ECC errors

#### Returns:

- [NVML\\_SUCCESS](#) if *eccCounts* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device*, *errorType* or *counterType* is invalid, or *eccCounts* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlDeviceClearEccErrorCounts\(\)](#)

#### 6.11.2.22 `nvmlReturn_t DECLDIR nvmlDeviceGetDisplayActive (nvmlDevice_t device, nvmlEnableState_t * isActive)`

Retrieves the display active state for the device.

For all products.

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

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

See [nvmlEnableState\\_t](#) for details on allowed modes.

#### Parameters:

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

***isActive*** Reference in which to return the display active state

#### Returns:

- [NVML\\_SUCCESS](#) if *isActive* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *isActive* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

Retrieves the display mode for the device.

For all products.

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

See [nvmlEnableState\\_t](#) for details on allowed modes.

#### Parameters:

- device* The identifier of the target device
- display* Reference in which to return the display mode

#### Returns:

- [NVML\\_SUCCESS](#) if *display* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *display* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

Retrieves the current and pending driver model for the device.

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

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

See [nvmlDriverModel\\_t](#) for details on available driver models.

#### Parameters:

- device* The identifier of the target device
- current* Reference in which to return the current driver model
- pending* Reference in which to return the pending driver model

#### Returns:

- [NVML\\_SUCCESS](#) if either *current* and/or *pending* have been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or both *current* and *pending* are NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the platform is not windows
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlDeviceSetDriverModel\(\)](#)

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

Retrieves the current and pending ECC modes for the device.

For Fermi <sup>TM</sup> or newer fully supported devices. Only applicable to devices with ECC. Requires `NVML_INFOROM_ECC` version 1.0 or higher.

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

See `nvmlEnableState_t` for details on allowed modes.

#### Parameters:

- device* The identifier of the target device
- current* Reference in which to return the current ECC mode
- pending* Reference in which to return the pending ECC mode

#### Returns:

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

#### See also:

`nvmlDeviceSetEccMode()`

### 6.11.2.26 `nvmlReturn_t DECLDIR nvmlDeviceGetEncoderUtilization (nvmlDevice_t device, unsigned int * utilization, unsigned int * samplingPeriodUs)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

#### Parameters:

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

#### Returns:

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

### 6.11.2.27 `nvmlReturn_t DECLDIR nvmlDeviceGetEnforcedPowerLimit (nvmlDevice_t device, unsigned int * limit)`

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

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

For Kepler <sup>TM</sup> or newer fully supported devices.

#### Parameters:

*device* The device to communicate with

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

#### Returns:

- [NVML\\_SUCCESS](#) if *limit* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *limit* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.28 `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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.29 `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 <sup>TM</sup>products from the Kepler family. Modes [NVML\\_GOM\\_LOW\\_DP](#) and [NVML\\_GOM\\_ALL\\_ON](#) are supported on fully supported GeForce products. Not supported on Quadro <sup>®</sup>and Tesla <sup>TM</sup>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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlGpuOperationMode\\_t](#)  
[nvmlDeviceSetGpuOperationMode](#)

### 6.11.2.30 `nvmlReturn_t DECLDIR nvmlDeviceGetGraphicsRunningProcesses (nvmlDevice_t device, unsigned int * infoCount, nvmlProcessInfo_t * infos)`

Get information about processes with a graphics context on a device

For Kepler <sup>TM</sup>or newer fully supported devices.

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

To query the current number of running graphics processes, call this function with *\*infoCount* = 0. The return code will be [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#), or [NVML\\_SUCCESS](#) if none are running. For this call *infos* is allowed to be NULL.

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

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

#### Parameters:

- device* The identifier of the target device
- infoCount* Reference in which to provide the *infos* array size, and to return the number of returned elements
- infos* Reference in which to return the process information

#### Returns:

- [NVML\\_SUCCESS](#) if *infoCount* and *infos* have been populated



- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *infoCount* indicates that the *infos* array is too small *infoCount* will contain minimal amount of space necessary for the call to complete
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, either of *infoCount* or *infos* is NULL
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlSystemGetProcessName](#)

#### 6.11.2.31 `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 UUID. See [nvmlDeviceGetHandleByUUID\(\)](#) and [nvmlDeviceGetHandleByPciBusId\(\)](#).

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

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

- The target GPU is an SLI slave

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

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

#### Parameters:

*index* The index of the target GPU,  $\geq 0$  and  $< accessibleDevices$

*device* Reference in which to return the device handle

#### Returns:

- [NVML\\_SUCCESS](#) if *device* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *index* is invalid or *device* is NULL
- [NVML\\_ERROR\\_INSUFFICIENT\\_POWER](#) if any attached devices have improperly attached external power cables
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to talk to this device

- [NVML\\_ERROR\\_IRQ\\_ISSUE](#) if NVIDIA kernel detected an interrupt issue with the attached GPUs
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceGetIndex](#)  
[nvmlDeviceGetCount](#)

#### 6.11.2.32 `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()`.

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

- The target GPU is an SLI slave

**Note:**

NVML 4.304 and older version of `nvmlDeviceGetHandleByPciBusId_v1` returns `NVML_ERROR_NOT_FOUND` instead of `NVML_ERROR_NO_PERMISSION`.

**Parameters:**

*pciBusId* The PCI bus id of the target GPU

*device* Reference in which to return the device handle

**Returns:**

- [NVML\\_SUCCESS](#) if *device* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *pciBusId* is invalid or *device* is NULL
- [NVML\\_ERROR\\_NOT\\_FOUND](#) if *pciBusId* does not match a valid device on the system
- [NVML\\_ERROR\\_INSUFFICIENT\\_POWER](#) if the attached device has improperly attached external power cables
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to talk to this device
- [NVML\\_ERROR\\_IRQ\\_ISSUE](#) if NVIDIA kernel detected an interrupt issue with the attached GPUs
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

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

For Fermi™ or newer fully supported devices.

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

**Deprecated**

Since more than one GPU can exist on a single board this function is deprecated in favor of [nvmlDeviceGetHandleByUUID](#). For dual GPU boards this function will return NVML\_ERROR\_INVALID\_ARGUMENT.

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

**Parameters:**

*serial* The board serial number of the target GPU  
*device* Reference in which to return the device handle

**Returns:**

- [NVML\\_SUCCESS](#) if *device* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *serial* is invalid, *device* is NULL or more than one device has the same serial (dual GPU boards)
- [NVML\\_ERROR\\_NOT\\_FOUND](#) if *serial* does not match a valid device on the system
- [NVML\\_ERROR\\_INSUFFICIENT\\_POWER](#) if any attached devices have improperly attached external power cables
- [NVML\\_ERROR\\_IRQ\\_ISSUE](#) if NVIDIA kernel detected an interrupt issue with the attached GPUs
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if any GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[nvmlDeviceGetSerial](#)  
[nvmlDeviceGetHandleByUUID](#)

#### 6.11.2.34 `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

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

**Returns:**

- [NVML\\_SUCCESS](#) if *device* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *uuid* is invalid or *device* is null
- [NVML\\_ERROR\\_NOT\\_FOUND](#) if *uuid* does not match a valid device on the system

- [NVML\\_ERROR\\_INSUFFICIENT\\_POWER](#) if any attached devices have improperly attached external power cables
- [NVML\\_ERROR\\_IRQ\\_ISSUE](#) if NVIDIA kernel detected an interrupt issue with the attached GPUs
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if any GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceGetUUID](#)

#### 6.11.2.35 `nvmlReturn_t DECLDIR nvmlDeviceGetIndex (nvmlDevice_t device, unsigned int * index)`

Retrieves the NVML index of this device.

For all products.

Valid indices are derived from the *accessibleDevices* count returned by [nvmlDeviceGetCount\(\)](#). For example, if *accessibleDevices* is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

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

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

**Parameters:**

*device* The identifier of the target device

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

**Returns:**

- [NVML\\_SUCCESS](#) if *index* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, or *index* is NULL
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceGetHandleByIndex\(\)](#)

[nvmlDeviceGetCount\(\)](#)

#### 6.11.2.36 `nvmlReturn_t DECLDIR nvmlDeviceGetInforomConfigurationChecksum (nvmlDevice_t device, unsigned int * checksum)`

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

For all products with an inforom.

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

**Parameters:**

- device* The identifier of the target device
- checksum* Reference in which to return the infoROM configuration checksum

**Returns:**

- [NVML\\_SUCCESS](#) if *checksum* has been set
- [NVML\\_ERROR\\_CORRUPTED\\_INFOROM](#) if the device's checksum couldn't be retrieved due to infoROM corruption
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *checksum* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.37 `nvmlReturn_t DECLDIR nvmlDeviceGetInforomImageVersion (nvmlDevice_t device, char * version, unsigned int length)`

Retrieves the global infoROM image version

For all products with an inforom.

Image version just like VBIOS version uniquely describes the exact version of the infoROM flashed on the board in contrast to infoROM object version which is only an indicator of supported features. Version string will not exceed 16 characters in length (including the NULL terminator). See [nvmlConstants::NVML\\_DEVICE\\_INFOROM\\_VERSION\\_BUFFER\\_SIZE](#).

**Parameters:**

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

**Returns:**

- [NVML\\_SUCCESS](#) if *version* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *version* is NULL
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *length* is too small
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not have an infoROM
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[nvmlDeviceGetInforomVersion](#)

### 6.11.2.38 `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 all products with an inforom.

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

See [nvmlInforomObject\\_t](#) for details on the available infoROM objects.

#### Parameters:

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

#### Returns:

- [NVML\\_SUCCESS](#) if *version* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *version* is NULL
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *length* is too small
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not have an infoROM
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlDeviceGetInforomImageVersion](#)

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

Retrieves the maximum clock speeds for the device.

For Fermi <sup>TM</sup> or newer fully supported devices.

See [nvmlClockType\\_t](#) for details on available clock information.

#### Note:

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

#### Parameters:

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

**Returns:**

- [NVML\\_SUCCESS](#) if *clock* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *clock* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device cannot report the specified clock
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.40 `nvmlReturn_t DECLDIR nvmlDeviceGetMaxCustomerBoostClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int * clockMHz)`

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

For newer than Maxwell <sup>TM</sup>fully supported devices.

**Parameters:**

*device* The identifier of the target device

*clockType* Identify which clock domain to query

*clockMHz* Reference in which to return the clock in MHz

**Returns:**

- [NVML\\_SUCCESS](#) if *clockMHz* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *clockMHz* is NULL or *clockType* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device or the *clockType* on this device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.41 `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 Fermi <sup>TM</sup>or newer fully supported devices.

**Parameters:**

*device* The identifier of the target device

*maxLinkGen* Reference in which to return the max PCIe link generation

**Returns:**

- [NVML\\_SUCCESS](#) if *maxLinkGen* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized

- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *maxLinkGen* is null
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if PCIe link information is not available
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.42 `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 Fermi <sup>TM</sup> or newer fully supported devices.

##### Parameters:

*device* The identifier of the target device

*maxLinkWidth* Reference in which to return the max PCIe link generation

##### Returns:

- [NVML\\_SUCCESS](#) if *maxLinkWidth* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *maxLinkWidth* is null
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if PCIe link information is not available
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.43 `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 Fermi <sup>TM</sup> or newer fully supported devices. Requires *NVML\_INFOROM\_ECC* version 2.0 or higher to report aggregate location-based memory error counts. Requires *NVML\_INFOROM\_ECC* version 1.0 or higher to report all other memory error counts.

Only applicable to devices with ECC.

Requires ECC Mode to be enabled.

See [nvmlMemoryErrorType\\_t](#) for a description of available memory error types.

See [nvmlEccCounterType\\_t](#) for a description of available counter types.

See [nvmlMemoryLocation\\_t](#) for a description of available counter locations.

##### 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*, *bitType*, *counterType* or *locationType* is invalid, or *count* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support ECC error reporting in the specified memory
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.44 `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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.45 `nvmlReturn_t DECLDIR nvmlDeviceGetMinorNumber (nvmlDevice_t device, unsigned int * minorNumber)`

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

For all products. Supported only for Linux

**Parameters:**

*device* The identifier of the target device

*minorNumber* Reference in which to return the minor number for the device

**Returns:**

- [NVML\\_SUCCESS](#) if the minor number is successfully retrieved
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *minorNumber* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this query is not supported by the device
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.46 `nvmlReturn_t DECLDIR nvmlDeviceGetMultiGpuBoard (nvmlDevice_t device, unsigned int * multiGpuBool)`

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

For Fermi <sup>TM</sup> or newer fully supported devices.

**Parameters:**

*device* The identifier of the target device

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

**Returns:**

- [NVML\\_SUCCESS](#) if *multiGpuBool* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *multiGpuBool* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.47 `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 <sup>TM</sup>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
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**6.11.2.48 nvmlReturn\_t DECLDIR nvmlDeviceGetPcieReplayCounter (nvmlDevice\_t device, unsigned int \* value)**

Retrieve the PCIe replay counter.

For Kepler <sup>TM</sup> or newer fully supported devices.

**Parameters:**

*device* The identifier of the target device

*value* Reference in which to return the counter's value

**Returns:**

- [NVML\\_SUCCESS](#) if *value* and *rollover* have been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, or *value* or *rollover* are NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**6.11.2.49 nvmlReturn\_t DECLDIR nvmlDeviceGetPcieThroughput (nvmlDevice\_t device, nvmlPcieUtilCounter\_t counter, unsigned int \* value)**

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

For Maxwell <sup>TM</sup> or newer fully supported devices.

This method is not supported on virtualized GPU environments.

**Parameters:**

*device* The identifier of the target device

*counter* The specific counter that should be queried [nvmlPcieUtilCounter\\_t](#)

*value* Reference in which to return throughput in KB/s

**Returns:**

- [NVML\\_SUCCESS](#) if *value* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* or *counter* is invalid, or *value* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**6.11.2.50 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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**6.11.2.51 nvmlReturn\_t DECLDIR nvmlDeviceGetPerformanceState (nvmlDevice\_t device, nvmlPstates\_t \* pState)**

Retrieves the current performance state for the device.

For Fermi <sup>TM</sup> or newer fully supported devices.

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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

Retrieves the persistence mode associated with this device.

For all products. For Linux only.

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

See [nvmlEnableState\\_t](#) for details on allowed modes.

##### Parameters:

*device* The identifier of the target device

*mode* Reference in which to return the current driver persistence mode

##### Returns:

- [NVML\\_SUCCESS](#) if *mode* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *mode* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

##### See also:

[nvmlDeviceSetPersistenceMode\(\)](#)

#### 6.11.2.53 `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 Kepler <sup>TM</sup> or newer fully supported devices.

##### Parameters:

*device* The identifier of the target device

*defaultLimit* Reference in which to return the default power management limit in milliwatts

##### Returns:

- [NVML\\_SUCCESS](#) if *defaultLimit* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *defaultLimit* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

Retrieves the power management limit associated with this device.

For Fermi <sup>TM</sup> or newer fully supported devices.

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

This reading is only available if power management mode is supported. See [nvmlDeviceGetPowerManagementMode](#).

##### Parameters:

*device* The identifier of the target device

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

##### Returns:

- [NVML\\_SUCCESS](#) if *limit* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *limit* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.55 `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 Kepler <sup>TM</sup> or newer fully supported devices.

##### Parameters:

*device* The identifier of the target device

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

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

##### Returns:

- [NVML\\_SUCCESS](#) if *minLimit* and *maxLimit* have been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *minLimit* or *maxLimit* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

##### See also:

[nvmlDeviceSetPowerManagementLimit](#)

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

This API has been deprecated.

Retrieves the power management mode associated with this device.

For products from the Fermi family.

- Requires `NVML_INFOROM_POWER` version 3.0 or higher.

For from the Kepler or newer families.

- Does not require `NVML_INFOROM_POWER` object.

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

See `nvmlEnableState_t` for details on allowed modes.

#### Parameters:

*device* The identifier of the target device

*mode* Reference in which to return the current power management mode

#### Returns:

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

### 6.11.2.57 `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 Fermi <sup>TM</sup> or newer fully supported devices.

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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.58 `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 Fermi <sup>TM</sup> or newer fully supported devices.

On Fermi and Kepler GPUs the reading is accurate to within +/- 5% of current power draw.

It is only available if power management mode is supported. See [nvmlDeviceGetPowerManagementMode](#).

##### Parameters:

*device* The identifier of the target device

*power* Reference in which to return the power usage information

##### Returns:

- [NVML\\_SUCCESS](#) if *power* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *power* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support power readings
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.59 `nvmlReturn_t DECLDIR nvmlDeviceGetRetiredPages (nvmlDevice_t device, nvmlPageRetirementCause_t cause, unsigned int * pageCount, unsigned long long * addresses)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

##### Parameters:

*device* The identifier of the target device

*cause* Filter page addresses by cause of retirement

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

*addresses* Buffer to write the page addresses into

##### Returns:

- [NVML\\_SUCCESS](#) if *pageCount* was populated and *addresses* was filled
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *pageCount* indicates the buffer is not large enough to store all the matching page addresses. *pageCount* is set to the needed size.



- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, *pageCount* is NULL, *cause* is invalid, or *addresses* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.60 `nvmlReturn_t DECLDIR nvmlDeviceGetRetiredPagesPendingStatus (nvmlDevice_t device, nvmlEnableState_t * isPending)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

##### Parameters:

*device* The identifier of the target device

*isPending* Reference in which to return the pending status

##### Returns:

- [NVML\\_SUCCESS](#) if *isPending* was populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *isPending* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

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

Gets recent samples for the GPU.

For Kepler <sup>TM</sup> or newer fully supported devices.

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

Power, Utilization and Clock samples are returned as type "unsigned int" for the union [nvmlValue\\_t](#).

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

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

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

**Parameters:**

*device* The identifier for the target device

*type* Type of sampling event

*lastSeenTimeStamp* Return only samples with timestamp greater than lastSeenTimeStamp.

*sampleValType* Output parameter to represent the type of sample value as described in `nvmlSampleVal_t`

*sampleCount* Reference to provide the number of elements which can be queried in samples array

*samples* Reference in which samples are returned

**Returns:**

- [NVML\\_SUCCESS](#) if samples are successfully retrieved
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, *samplesCount* is NULL or reference to *sampleCount* is 0 for non null *samples*
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this query is not supported by the device
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_NOT\\_FOUND](#) if sample entries are not found
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.62 `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 all products with an inform.

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
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.63 `nvmlReturn_t DECLDIR nvmlDeviceGetSupportedClocksThrottleReasons (nvmlDevice_t device, unsigned long long * supportedClocksThrottleReasons)`

Retrieves bitmask of supported clocks throttle reasons that can be returned by [nvmlDeviceGetCurrentClocksThrottleReasons](#)

For all fully supported products.

This method is not supported on virtualized GPU environments.

#### Parameters:

*device* The identifier of the target device

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

#### Returns:

- [NVML\\_SUCCESS](#) if *supportedClocksThrottleReasons* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *supportedClocksThrottleReasons* is NULL
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[NvmlClocksThrottleReasons](#)

[nvmlDeviceGetCurrentClocksThrottleReasons](#)

### 6.11.2.64 `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 Kepler™ or newer fully supported devices.

#### Parameters:

*device* The identifier of the target device

*memoryClockMHz* Memory clock for which to return possible graphics clocks

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

*clocksMHz* Reference in which to return the clocks in MHz

#### Returns:

- [NVML\\_SUCCESS](#) if *count* and *clocksMHz* have been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_NOT\\_FOUND](#) if the specified *memoryClockMHz* is not a supported frequency
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *clock* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *count* is too small
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceSetApplicationsClocks](#)  
[nvmlDeviceGetSupportedMemoryClocks](#)

#### 6.11.2.65 `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 Kepler™ or newer fully supported devices.

**Parameters:**

*device* The identifier of the target device  
*count* Reference in which to provide the *clocksMHz* array size, and to return the number of elements  
*clocksMHz* Reference in which to return the clock in MHz

**Returns:**

- [NVML\\_SUCCESS](#) if *count* and *clocksMHz* have been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *count* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_INSUFFICIENT\\_SIZE](#) if *count* is too small (*count* is set to the number of required elements)
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceSetApplicationsClocks](#)  
[nvmlDeviceGetSupportedGraphicsClocks](#)

#### 6.11.2.66 `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 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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.67 `nvmlReturn_t DECLDIR nvmlDeviceGetTemperatureThreshold (nvmlDevice_t device, nvmlTemperatureThresholds_t thresholdType, unsigned int * temp)`

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

For Kepler <sup>TM</sup> or newer fully supported devices.

See [nvmlTemperatureThresholds\\_t](#) for details on available temperature thresholds.

##### Parameters:

*device* The identifier of the target device

*thresholdType* The type of threshold value queried

*temp* Reference in which to return the temperature reading

##### Returns:

- [NVML\\_SUCCESS](#) if *temp* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, *thresholdType* is invalid or *temp* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not have a temperature sensor or is unsupported
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.68 `nvmlReturn_t DECLDIR nvmlDeviceGetTopologyCommonAncestor (nvmlDevice_t device1, nvmlDevice_t device2, nvmlGpuTopologyLevel_t * pathInfo)`

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

##### Parameters:

*device1* The identifier of the first device

*device2* The identifier of the second device

*pathInfo* A [nvmlGpuTopologyLevel\\_t](#) that gives the path type

##### Returns:

- [NVML\\_SUCCESS](#) if *pathInfo* has been set
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device1*, or *device2* is invalid, or *pathInfo* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device or OS does not support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) an error has occurred in underlying topology discovery

### 6.11.2.69 `nvmlReturn_t DECLDIR nvmlDeviceGetTopologyNearestGpus (nvmlDevice_t device, nvmlGpuTopologyLevel_t level, unsigned int * count, nvmlDevice_t * deviceArray)`

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

#### Parameters:

- device* The identifier of the first device
- level* The `nvmlGpuTopologyLevel_t` level to search for other GPUs
- count* When zero, is set to the number of matching GPUs such that *deviceArray* can be malloc'd. When non-zero, *deviceArray* will be filled with *count* number of device handles.
- deviceArray* An array of device handles for GPUs found at *level*

#### Returns:

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

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

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

See `nvmlMemoryErrorType_t` for a description of available error types.

See `nvmlEccCounterType_t` for a description of available counter types.

#### Parameters:

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

#### Returns:

- `NVML_SUCCESS` if *eccCounts* has been set
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if *device*, *errorType* or *counterType* is invalid, or *eccCounts* is NULL
- `NVML_ERROR_NOT_SUPPORTED` if the device does not support this feature

- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceClearEccErrorCounts\(\)](#)

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

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

For Fermi™ or newer fully supported devices.

See [nvmlUtilization\\_t](#) for details on available utilization rates.

**Note:**

During driver initialization when ECC is enabled one can see high GPU and Memory Utilization readings. This is caused by ECC Memory Scrubbing mechanism that is performed during driver initialization.

**Parameters:**

*device* The identifier of the target device

*utilization* Reference in which to return the utilization information

**Returns:**

- [NVML\\_SUCCESS](#) if *utilization* has been populated
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *utilization* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.72 `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 products.

The UUID is a globally unique identifier. It is the only available identifier for pre-Fermi-architecture products. It does NOT correspond to any identifier printed on the board. It will not exceed 80 characters in length (including the NULL terminator). See [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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.73 `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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.74 `nvmlReturn_t DECLDIR nvmlDeviceGetViolationStatus (nvmlDevice_t device, nvmlPerfPolicyType_t perfPolicyType, nvmlViolationTime_t * violTime)`

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

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

Violation for thermal capping is not supported at this time.

For Kepler <sup>TM</sup> or newer fully supported devices.

**Parameters:**

- device* The identifier of the target device



**perfPolicyType** Represents Performance policy which can trigger GPU throttling

**violTime** Reference to which violation time related information is returned

**Returns:**

- [NVML\\_SUCCESS](#) if violation time is successfully retrieved
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid, *perfPolicyType* is invalid, or *violTime* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this query is not supported by the device
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible

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

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

For all fully supported products.

**Parameters:**

**device1** The first GPU device

**device2** The second GPU device

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

**Returns:**

- [NVML\\_SUCCESS](#) if *onSameBoard* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *dev1* or *dev2* are invalid or *onSameBoard* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this check is not supported by the device
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the either GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.76 `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 can be changed using [nvmlDeviceSetApplicationsClocks](#).

On Pascal and newer hardware, if clocks were previously locked with [nvmlDeviceSetApplicationsClocks](#), this call will unlock clocks. This returns clocks their default behavior of automatically boosting above base clocks as thermal limits allow.

**See also:**

[nvmlDeviceGetApplicationsClock](#)

[nvmlDeviceSetApplicationsClocks](#)

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

**Parameters:**

*device* The identifier of the target device

**Returns:**

- [NVML\\_SUCCESS](#) if new settings were successfully set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.77 [nvmlReturn\\_t DECLDIR nvmlDeviceSetAutoBoostedClocksEnabled](#) ([nvmlDevice\\_t device](#), [nvmlEnableState\\_t enabled](#))

Try to set the current state of Auto Boosted clocks on a device.

For Kepler <sup>TM</sup> or newer fully supported devices.

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

Non-root users may use this API by default but can be restricted by root from using this API by calling [nvmlDeviceSetAPIRestriction](#) with `apiType=NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS`. Note: Persistence Mode is required to modify current Auto Boost settings, therefore, it must be enabled.

On Pascal and newer hardware, Auto Boosted clocks are controlled through application clocks. Use [nvmlDeviceSetApplicationsClocks](#) and [nvmlDeviceResetApplicationsClocks](#) to control Auto Boost behavior.

**Parameters:**

*device* The identifier of the target device

*enabled* What state to try to set Auto Boosted clocks of the target device to

**Returns:**

- [NVML\\_SUCCESS](#) If the Auto Boosted clocks were successfully set to the state specified by *enabled*
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support Auto Boosted clocks
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.78 [nvmlReturn\\_t DECLDIR nvmlDeviceSetCpuAffinity](#) ([nvmlDevice\\_t device](#))

Sets the ideal affinity for the calling thread and device using the guidelines given in [nvmlDeviceGetCpuAffinity\(\)](#). Note, this is a change as of version 8.0. Older versions set the affinity for a calling process and all children. Currently supports up to 64 processors.

For Kepler <sup>TM</sup> or newer fully supported devices. Supported on Linux only.

**Parameters:**

*device* The identifier of the target device

**Returns:**

- [NVML\\_SUCCESS](#) if the calling process has been successfully bound
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.79 `nvmlReturn_t DECLDIR nvmlDeviceSetDefaultAutoBoostedClocksEnabled (nvmlDevice_t device, nvmlEnableState_t enabled, unsigned int flags)`

Try to set the default state of Auto Boosted clocks on a device. This is the default state that Auto Boosted clocks will return to when no compute running processes (e.g. CUDA application which have an active context) are running

For Kepler™ or newer non-GeForce fully supported devices and Maxwell or newer GeForce devices. Requires root/admin permissions.

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

On Pascal and newer hardware, Auto Boosted clocks are controlled through application clocks. Use [nvmlDeviceSetApplicationsClocks](#) and [nvmlDeviceResetApplicationsClocks](#) to control Auto Boost behavior.

**Parameters:**

*device* The identifier of the target device

*enabled* What state to try to set default Auto Boosted clocks of the target device to

*flags* Flags that change the default behavior. Currently Unused.

**Returns:**

- [NVML\\_SUCCESS](#) If the Auto Boosted clock's default state was successfully set to the state specified by *enabled*
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) If the calling user does not have permission to change Auto Boosted clock's default state.
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support Auto Boosted clocks
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.11.2.80 `nvmlReturn_t DECLDIR nvmlDeviceValidateInforom (nvmlDevice_t device)`

Reads the infoROM from the flash and verifies the checksums.

For all products with an inforom.

**Parameters:**

*device* The identifier of the target device

**Returns:**

- [NVML\\_SUCCESS](#) if infoROM is not corrupted
- [NVML\\_ERROR\\_CORRUPTED\\_INFOTROM](#) if the device's infoROM is corrupted
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.11.2.81 `nvmlReturn_t DECLDIR nvmlSystemGetTopologyGpuSet (unsigned int cpuNumber, unsigned int * count, nvmlDevice_t * deviceArray)`

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

**Parameters:**

*cpuNumber* The CPU number

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

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

**Returns:**

- [NVML\\_SUCCESS](#) if *deviceArray* or *count* (if initially zero) has been set
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *cpuNumber*, or *count* is invalid, or *deviceArray* is NULL with a non-zero *count*
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device or OS does not support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) an error has occurred in underlying topology discovery

## 6.12 Unit Commands

### Functions

- [nvmlReturn\\_t](#) DECLDIR [nvmlUnitSetLedState](#) (nvmlUnit\_t unit, [nvmlLedColor\\_t](#) color)

#### 6.12.1 Detailed Description

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

#### 6.12.2 Function Documentation

##### 6.12.2.1 [nvmlReturn\\_t](#) DECLDIR [nvmlUnitSetLedState](#) (nvmlUnit\_t *unit*, [nvmlLedColor\\_t](#) *color*)

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

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

This operation takes effect immediately.

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

See [nvmlLedColor\\_t](#) for available colors.

##### Parameters:

*unit* The identifier of the target unit

*color* The target LED color

##### Returns:

- [NVML\\_SUCCESS](#) if the LED color has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *unit* or *color* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if this is not an S-class product
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

##### See also:

[nvmlUnitGetLedState\(\)](#)

## 6.13 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)
- [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceSetAPIRestriction](#) ([nvmlDevice\\_t](#) device, [nvmlRestrictedAPI\\_t](#) apiType, [nvmlEnableState\\_t](#) isRestricted)

### 6.13.1 Detailed Description

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

### 6.13.2 Function Documentation

#### 6.13.2.1 [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceClearEccErrorCounts](#) ([nvmlDevice\\_t](#) *device*, [nvmlEccCounterType\\_t](#) *counterType*)

Clear the ECC error and other memory error counts for the device.

For Kepler™ or newer fully supported devices. Only applicable to devices with ECC. Requires *NVML\_INFOROM\_ECC* version 2.0 or higher to clear aggregate location-based ECC counts. Requires *NVML\_INFOROM\_ECC* version 1.0 or higher to clear all other ECC counts. Requires root/admin permissions. Requires ECC Mode to be enabled.

Sets all of the specified ECC counters to 0, including both detailed and total counts.

This operation takes effect immediately.

See [nvmlMemoryErrorType\\_t](#) for details on available counter types.

#### Parameters:

*device* The identifier of the target device

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

#### Returns:

- [NVML\\_SUCCESS](#) if the error counts were cleared
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *counterType* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation

- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

- [nvmlDeviceGetDetailedEccErrors\(\)](#)
- [nvmlDeviceGetTotalEccErrors\(\)](#)

#### 6.13.2.2 `nvmlReturn_t DECLDIR nvmlDeviceSetAPIRestriction (nvmlDevice_t device, nvmlRestrictedAPI_t apiType, nvmlEnableState_t isRestricted)`

Changes the root/admin restrictions on certain APIs. See *nvmlRestrictedAPI\_t* for the list of supported APIs. This method can be used by a root/admin user to give non-root/admin access to certain otherwise-restricted APIs. The new setting lasts for the lifetime of the NVIDIA driver; it is not persistent. See *nvmlDeviceGetAPIRestriction* to query the current restriction settings.

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

Parameters:

*device* The identifier of the target device  
*apiType* Target API type for this operation  
*isRestricted* The target restriction

Returns:

- [NVML\\_SUCCESS](#) if *isRestricted* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *apiType* incorrect
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support changing API restrictions or the device does not support the feature that api restrictions are being set for (E.G. Enabling/disabling auto boosted clocks is not supported by the device)
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlRestrictedAPI\\_t](#)

#### 6.13.2.3 `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.

On Pascal and newer hardware, this will automatically disable automatic boosting of clocks.

On K80 and newer Kepler and Maxwell GPUs, users desiring fixed performance should also call [nvmlDeviceSetAutoBoostedClocksEnabled](#) to prevent clocks from automatically boosting above the clock value being set.

For Kepler <sup>TM</sup> or newer non-GeForce fully supported devices and Maxwell or newer GeForce devices. Requires root/admin permissions.

See [nvmlDeviceGetSupportedMemoryClocks](#) and [nvmlDeviceGetSupportedGraphicsClocks](#) for details on how to list available clocks combinations.

After system reboot or driver reload applications clocks go back to their default value. See [nvmlDeviceResetApplicationsClocks](#).

#### Parameters:

- device* The identifier of the target device
- memClockMHz* Requested memory clock in MHz
- graphicsClockMHz* Requested graphics clock in MHz

#### Returns:

- [NVML\\_SUCCESS](#) if new settings were successfully set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *memClockMHz* and *graphicsClockMHz* is not a valid clock combination
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.13.2.4 `nvmlReturn_t DECLDIR nvmlDeviceSetComputeMode (nvmlDevice_t device, nvmlComputeMode_t mode)`

Set the compute mode for the device.

For all products. Requires root/admin permissions.

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

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

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

See [nvmlComputeMode\\_t](#) 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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceGetComputeMode\(\)](#)

#### 6.13.2.5 `nvmlReturn_t DECLDIR nvmlDeviceSetDriverModel (nvmlDevice_t device, nvmlDriverModel_t driverModel, unsigned int flags)`

Set the driver model for the device.

For Fermi™ or newer fully supported devices. For windows only. Requires root/admin permissions.

On Windows platforms the device driver can run in either WDDM or WDM (TCC) mode. If a display is attached to the device it must run in WDDM mode.

It is possible to force the change to WDM (TCC) while the display is still attached with a force flag ([nvmlFlagForce](#)). This should only be done if the host is subsequently powered down and the display is detached from the device before the next reboot.

This operation takes effect after the next reboot.

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

Change driver model to WDDM is not supported when GPU doesn't support graphics acceleration or will not support it after reboot. See [nvmlDeviceSetGpuOperationMode](#).

See [nvmlDriverModel\\_t](#) for details on available driver models. See [nvmlFlagDefault](#) and [nvmlFlagForce](#)

#### Parameters:

*device* The identifier of the target device

*driverModel* The target driver model

*flags* Flags that change the default behavior

#### Returns:

- [NVML\\_SUCCESS](#) if the driver model has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *driverModel* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the platform is not windows or the device does not support this feature
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceGetDriverModel\(\)](#)

#### 6.13.2.6 `nvmlReturn_t DECLDIR nvmlDeviceSetEccMode (nvmlDevice_t device, nvmlEnableState_t ecc)`

Set the ECC mode for the device.

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

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

This operation takes effect after the next reboot.

See `nvmlEnableState_t` for details on available modes.

##### Parameters:

*device* The identifier of the target device

*ecc* The target ECC mode

##### Returns:

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

##### See also:

`nvmlDeviceGetEccMode()`

#### 6.13.2.7 `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 <sup>TM</sup> products from the Kepler family. Modes `NVML_GOM_LOW_DP` and `NVML_GOM_ALL_ON` are supported on fully supported GeForce products. Not supported on Quadro <sup>®</sup> and Tesla <sup>TM</sup> 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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlGpuOperationMode\\_t](#)  
[nvmlDeviceGetGpuOperationMode](#)

#### 6.13.2.8 `nvmlReturn_t DECLDIR nvmlDeviceSetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t mode)`

Set the persistence mode for the device.

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

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

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

See [nvmlEnableState\\_t](#) for available modes.

##### Parameters:

*device* The identifier of the target device

*mode* The target persistence mode

##### Returns:

- [NVML\\_SUCCESS](#) if the persistence mode was set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* is invalid or *mode* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device does not support this feature
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceGetPersistenceMode\(\)](#)

#### 6.13.2.9 `nvmlReturn_t DECLDIR nvmlDeviceSetPowerManagementLimit (nvmlDevice_t device, unsigned int limit)`

Set new power limit of this device.

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

See [nvmlDeviceGetPowerManagementLimitConstraints](#) to check the allowed ranges of values.

**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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[nvmlDeviceGetPowerManagementLimitConstraints](#)  
[nvmlDeviceGetPowerManagementDefaultLimit](#)

## 6.14 NvLink Methods

### Functions

- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkState` (`nvmlDevice_t device`, `unsigned int link`, `nvmlEnableState_t *isActive`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkVersion` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int *version`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkCapability` (`nvmlDevice_t device`, `unsigned int link`, `nvmlNvLinkCapability_t capability`, `unsigned int *capResult`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkRemotePciInfo` (`nvmlDevice_t device`, `unsigned int link`, `nvmlPciInfo_t *pci`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkErrorCounter` (`nvmlDevice_t device`, `unsigned int link`, `nvmlNvLinkErrorCounter_t counter`, `unsigned long long *counterValue`)
- `nvmlReturn_t DECLDIR nvmlDeviceResetNvLinkErrorCounters` (`nvmlDevice_t device`, `unsigned int link`)
- `nvmlReturn_t DECLDIR nvmlDeviceSetNvLinkUtilizationControl` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `nvmlNvLinkUtilizationControl_t *control`, `unsigned int reset`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkUtilizationControl` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `nvmlNvLinkUtilizationControl_t *control`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkUtilizationCounter` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `unsigned long long *rxcounter`, `unsigned long long *txcounter`)
- `nvmlReturn_t DECLDIR nvmlDeviceFreezeNvLinkUtilizationCounter` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `nvmlEnableState_t freeze`)
- `nvmlReturn_t DECLDIR nvmlDeviceResetNvLinkUtilizationCounter` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`)

### 6.14.1 Detailed Description

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

### 6.14.2 Function Documentation

#### 6.14.2.1 `nvmlReturn_t DECLDIR nvmlDeviceFreezeNvLinkUtilizationCounter` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `nvmlEnableState_t freeze`)

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

For newer than Maxwell <sup>TM</sup>fully supported devices.

#### Parameters:

*device* The identifier of the target device

*link* Specifies the NvLink link to be queried

*counter* Specifies the counter that should be frozen (0 or 1).

*freeze* NVML\_FEATURE\_ENABLED = freeze the receive and transmit counters NVML\_FEATURE\_DISABLED = unfreeze the receive and transmit counters

#### Returns:

- `NVML_SUCCESS` if counters were successfully frozen or unfrozen
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized

- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device*, *link*, *counter*, or *freeze* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.2 `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkCapability (nvmlDevice_t device, unsigned int link, nvmlNvLinkCapability_t capability, unsigned int * capResult)`

Retrieves the requested capability from the device's NvLink for the link specified. Please refer to the `nvmlNvLinkCapability_t` structure for the specific caps that can be queried. The return value should be treated as a boolean.

For newer than Maxwell™ fully supported devices.

##### Parameters:

***device*** The identifier of the target device  
***link*** Specifies the NvLink link to be queried  
***capability*** Specifies the `nvmlNvLinkCapability_t` to be queried  
***capResult*** A boolean for the queried capability indicating that feature is available

##### Returns:

- [NVML\\_SUCCESS](#) if *capResult* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device*, *link*, or *capability* is invalid or *capResult* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.3 `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkErrorCounter (nvmlDevice_t device, unsigned int link, nvmlNvLinkErrorCounter_t counter, unsigned long long * counterValue)`

Retrieves the specified error counter value. Please refer to `nvmlNvLinkErrorCounter_t` for error counters that are available.

For newer than Maxwell™ fully supported devices.

##### Parameters:

***device*** The identifier of the target device  
***link*** Specifies the NvLink link to be queried  
***counter*** Specifies the NvLink counter to be queried  
***counterValue*** Returned counter value

##### Returns:

- [NVML\\_SUCCESS](#) if *counter* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device*, *link*, or *counter* is invalid or *counterValue* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.4 `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkRemotePciInfo (nvmlDevice_t device, unsigned int link, nvmlPciInfo_t * pci)`

Retrieves the PCI information for the remote node on a NvLink link Note: pciSubSystemId is not filled in this function and is indeterminate

For newer than Maxwell <sup>TM</sup>fully supported devices.

##### Parameters:

- device* The identifier of the target device
- link* Specifies the NvLink link to be queried
- pci* [nvmlPciInfo\\_t](#) of the remote node for the specified link

##### Returns:

- [NVML\\_SUCCESS](#) if *pci* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* or *link* is invalid or *pci* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.5 `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkState (nvmlDevice_t device, unsigned int link, nvmlEnableState_t * isActive)`

Retrieves the state of the device's NvLink for the link specified

For newer than Maxwell <sup>TM</sup>fully supported devices.

##### Parameters:

- device* The identifier of the target device
- link* Specifies the NvLink link to be queried
- isActive* [nvmlEnableState\\_t](#) where [NVML\\_FEATURE\\_ENABLED](#) indicates that the link is active and [NVML\\_FEATURE\\_DISABLED](#) indicates it is inactive

##### Returns:

- [NVML\\_SUCCESS](#) if *isActive* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* or *link* is invalid or *isActive* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.6 `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkUtilizationControl (nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlNvLinkUtilizationControl_t * control)`

Get the NVLINK utilization counter control information for the specified counter, 0 or 1. Please refer to [nvmlNvLinkUtilizationControl\\_t](#) for the structure definition

For newer than Maxwell <sup>TM</sup>fully supported devices.

**Parameters:**

- device* The identifier of the target device
- counter* Specifies the counter that should be set (0 or 1).
- link* Specifies the NvLink link to be queried
- control* A reference to the [nvmlNvLinkUtilizationControl\\_t](#) to place information

**Returns:**

- [NVML\\_SUCCESS](#) if the control has been set successfully
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device*, *counter*, *link*, or *control* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.7 `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkUtilizationCounter (nvmlDevice_t device, unsigned int link, unsigned int counter, unsigned long long * rxcounter, unsigned long long * txcounter)`

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

For newer than Maxwell <sup>TM</sup>fully supported devices.

**Parameters:**

- device* The identifier of the target device
- link* Specifies the NvLink link to be queried
- counter* Specifies the counter that should be read (0 or 1).
- rxcounter* Receive counter return value
- txcounter* Transmit counter return value

**Returns:**

- [NVML\\_SUCCESS](#) if *rxcounter* and *txcounter* have been successfully set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device*, *counter*, or *link* is invalid or *rxcounter* or *txcounter* are NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.8 `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkVersion (nvmlDevice_t device, unsigned int link, unsigned int * version)`

Retrieves the version of the device's NvLink for the link specified

For newer than Maxwell <sup>TM</sup>fully supported devices.

**Parameters:**

- device* The identifier of the target device



*link* Specifies the NvLink link to be queried

*version* Requested NvLink version

**Returns:**

- [NVML\\_SUCCESS](#) if *version* has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* or *link* is invalid or *version* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.9 `nvmlReturn_t DECLDIR nvmlDeviceResetNvLinkErrorCounters (nvmlDevice_t device, unsigned int link)`

Resets all error counters to zero Please refer to *nvmlNvLinkErrorCounter\_t* for the list of error counters that are reset For newer than Maxwell <sup>TM</sup>fully supported devices.

**Parameters:**

*device* The identifier of the target device

*link* Specifies the NvLink link to be queried

**Returns:**

- [NVML\\_SUCCESS](#) if the reset is successful
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device* or *link* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.10 `nvmlReturn_t DECLDIR nvmlDeviceResetNvLinkUtilizationCounter (nvmlDevice_t device, unsigned int link, unsigned int counter)`

Reset the NVLINK utilization counters Both the receive and transmit counters are operated on by this function For newer than Maxwell <sup>TM</sup>fully supported devices.

**Parameters:**

*device* The identifier of the target device

*link* Specifies the NvLink link to be reset

*counter* Specifies the counter that should be reset (0 or 1)

**Returns:**

- [NVML\\_SUCCESS](#) if counters were successfully reset
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device*, *link*, or *counter* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### 6.14.2.11 `nvmlReturn_t DECLDIR nvmlDeviceSetNvLinkUtilizationControl (nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlNvLinkUtilizationControl_t * control, unsigned int reset)`

Set the NVLINK utilization counter control information for the specified counter, 0 or 1. Please refer to [nvmlNvLinkUtilizationControl\\_t](#) for the structure definition. Performs a reset of the counters if the reset parameter is non-zero.

For newer than Maxwell <sup>TM</sup>fully supported devices.

##### Parameters:

- device* The identifier of the target device
- counter* Specifies the counter that should be set (0 or 1).
- link* Specifies the NvLink link to be queried
- control* A reference to the [nvmlNvLinkUtilizationControl\\_t](#) to set
- reset* Resets the counters on set if non-zero

##### Returns:

- [NVML\\_SUCCESS](#) if the control has been set successfully
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *device*, *counter*, *link*, or *control* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

## 6.15 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 timeouts)
- [nvmlReturn\\_t](#) DECLDIR [nvmlEventSetFree](#) ([nvmlEventSet\\_t](#) set)

#### 6.15.1 Detailed Description

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

#### 6.15.2 Typedef Documentation

##### 6.15.2.1 typedef struct nvmlEventSet\_st\* nvmlEventSet\_t

Handle to an event set

#### 6.15.3 Function Documentation

##### 6.15.3.1 [nvmlReturn\\_t](#) DECLDIR [nvmlDeviceGetSupportedEventTypes](#) ([nvmlDevice\\_t](#) *device*, unsigned long long \* *eventTypes*)

Returns information about events supported on device

For Fermi <sup>TM</sup> or newer fully supported devices.

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\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[Event Types](#)  
[nvmlDeviceRegisterEvents](#)

### 6.15.3.2 `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 Fermi <sup>TM</sup> or newer fully supported devices. Ecc events are available only on ECC enabled devices (see [nvmlDeviceGetTotalEccErrors](#)) Power capping events are available only on Power Management enabled devices (see [nvmlDeviceGetPowerManagementMode](#))

For Linux only.

**IMPORTANT:** Operations on *set* are not thread safe

This call starts recording of events on specific device. All events that occurred before this call are not recorded. Checking if some event occurred can be done with [nvmlEventSetWait](#)

If function reports NVML\_ERROR\_UNKNOWN, event set is in undefined state and should be freed. If function reports NVML\_ERROR\_NOT\_SUPPORTED, event set can still be used. None of the requested eventTypes are registered in that case.

**Parameters:**

*device* The identifier of the target device  
*eventTypes* Bitmask of [Event Types](#) to record  
*set* Set to which add new event types

**Returns:**

- [NVML\\_SUCCESS](#) if the event has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *eventTypes* is invalid or *set* is NULL
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the platform does not support this feature or some of requested event types
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[Event Types](#)  
[nvmlDeviceGetSupportedEventTypes](#)  
[nvmlEventSetWait](#)  
[nvmlEventSetFree](#)

### 6.15.3.3 `nvmlReturn_t DECLDIR nvmlEventSetCreate (nvmlEventSet_t * set)`

Create an empty set of events. Event set should be freed by [nvmlEventSetFree](#)

For Fermi <sup>TM</sup> or newer fully supported devices.

#### Parameters:

*set* Reference in which to return the event handle

#### Returns:

- [NVML\\_SUCCESS](#) if the event has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *set* is NULL
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlEventSetFree](#)

### 6.15.3.4 `nvmlReturn_t DECLDIR nvmlEventSetFree (nvmlEventSet_t set)`

Releases events in the set

For Fermi <sup>TM</sup> or newer fully supported devices.

#### Parameters:

*set* Reference to events to be released

#### Returns:

- [NVML\\_SUCCESS](#) if the event has been successfully released
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

#### See also:

[nvmlDeviceRegisterEvents](#)

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

Waits on events and delivers events

For Fermi <sup>TM</sup> or newer fully supported devices.

If some events are ready to be delivered at the time of the call, function returns immediately. If there are no events ready to be delivered, function sleeps till event arrives but not longer than specified timeout. This function in certain conditions can return before specified timeout passes (e.g. when interrupt arrives)

In case of xid error, the function returns the most recent xid error type seen by the system. If there are multiple xid errors generated before `nvmlEventSetWait` is invoked then the last seen xid error type is returned for all xid error events.

**Parameters:**

*set* Reference to set of events to wait on

*data* Reference in which to return event data

*timeoutms* Maximum amount of wait time in milliseconds for registered event

**Returns:**

- [NVML\\_SUCCESS](#) if the data has been set
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *data* is NULL
- [NVML\\_ERROR\\_TIMEOUT](#) if no event arrived in specified timeout or interrupt arrived
- [NVML\\_ERROR\\_GPU\\_IS\\_LOST](#) if a GPU has fallen off the bus or is otherwise inaccessible
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

**See also:**

[Event Types](#)

[nvmlDeviceRegisterEvents](#)

## 6.16 Drain states

### Functions

- `nvmlReturn_t` DECLDIR `nvmlDeviceModifyDrainState` (unsigned int `nvmlIndex`, `nvmlEnableState_t` `newState`)
- `nvmlReturn_t` DECLDIR `nvmlDeviceQueryDrainState` (unsigned int `nvmlIndex`, `nvmlEnableState_t` `*currentState`)
- `nvmlReturn_t` DECLDIR `nvmlDeviceRemoveGpu` (unsigned int `nvmlIndex`)
- `nvmlReturn_t` DECLDIR `nvmlDeviceDiscoverGpus` (`nvmlPciInfo_t` `*pciInfo`)

### 6.16.1 Detailed Description

This chapter describes methods that NVML can perform against each device to control their drain state and recognition by NVML and NVIDIA kernel driver. These methods can be used with out-of-band tools to power on/off GPUs, enable robust reset scenarios, etc.

### 6.16.2 Function Documentation

#### 6.16.2.1 `nvmlReturn_t` DECLDIR `nvmlDeviceDiscoverGpus` (`nvmlPciInfo_t` `*pciInfo`)

Request the OS and the NVIDIA kernel driver to rediscover a portion of the PCI subsystem looking for GPUs that were previously removed. The portion of the PCI tree can be narrowed by specifying a domain, bus, and device. If all are zeroes then the entire PCI tree will be searched. Please note that for long-running NVML processes the enumeration will change based on how many GPUs are discovered and where they are inserted in bus order.

In addition, all newly discovered GPUs will be initialized and their ECC scrubbed which may take several seconds per GPU. Also, all device handles are no longer guaranteed to be valid post discovery.

Must be run as administrator. For Linux only.

For newer than Maxwell <sup>TM</sup>fully supported devices. Some Kepler devices supported.

#### Parameters:

*pciInfo* The PCI tree to be searched. Only the domain, bus, and device fields are used in this call.

#### Returns:

- `NVML_SUCCESS` if counters were successfully reset
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if *pciInfo* is invalid
- `NVML_ERROR_NOT_SUPPORTED` if the operating system does not support this feature
- `NVML_ERROR_OPERATING_SYSTEM` if the operating system is denying this feature
- `NVML_ERROR_NO_PERMISSION` if the calling process has insufficient permissions to perform operation
- `NVML_ERROR_UNKNOWN` on any unexpected error

#### 6.16.2.2 `nvmlReturn_t` DECLDIR `nvmlDeviceModifyDrainState` (unsigned int `nvmlIndex`, `nvmlEnableState_t` `newState`)

Modify the drain state of a GPU. This method forces a GPU to no longer accept new incoming requests. Any new NVML process will see a gap in the enumeration where this GPU should exist as any call to that GPU outside of the drain state APIs will fail. Must be called as administrator. For Linux only.

For newer than Maxwell™ fully supported devices. Some Kepler devices supported.

**Parameters:**

- nvmlIndex* The ID of the target device
- newState* The drain state that should be entered, see [nvmlEnableState\\_t](#)

**Returns:**

- [NVML\\_SUCCESS](#) if counters were successfully reset
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *nvmlIndex* or *newState* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_NO\\_PERMISSION](#) if the calling process has insufficient permissions to perform operation
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.16.2.3 `nvmlReturn_t DECLDIR nvmlDeviceQueryDrainState (unsigned int nvmlIndex, nvmlEnableState_t * currentState)`

Query the drain state of a GPU. This method is used to check if a GPU is in a currently draining state. For Linux only. For newer than Maxwell™ fully supported devices. Some Kepler devices supported.

**Parameters:**

- nvmlIndex* The ID of the target device
- currentState* The current drain state for this GPU, see [nvmlEnableState\\_t](#)

**Returns:**

- [NVML\\_SUCCESS](#) if counters were successfully reset
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *nvmlIndex* or *currentState* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_UNKNOWN](#) on any unexpected error

### 6.16.2.4 `nvmlReturn_t DECLDIR nvmlDeviceRemoveGpu (unsigned int nvmlIndex)`

This method will remove the specified GPU from the view of both NVML and the NVIDIA kernel driver as long as no other processes are attached. If other processes are attached, this call will return [NVML\\_ERROR\\_IN\\_USE](#) and the GPU will be returned to its original "draining" state. Note: the only situation where a process can still be attached after [nvmlDeviceModifyDrainState\(\)](#) is called to initiate the draining state is if that process was using, and is still using, a GPU before the call was made. Also note, persistence mode counts as an attachment to the GPU thus it must be disabled prior to this call.

For long-running NVML processes please note that this will change the enumeration of current GPUs. For example, if there are four GPUs present and GPU1 is removed, the new enumeration will be 0-2. Also, device handles after the removed GPU will not be valid and must be re-established. Must be run as administrator. For Linux only.

For newer than Maxwell™ fully supported devices. Some Kepler devices supported.



**Parameters:**

*nvmlIndex* The ID of the target device

**Returns:**

- [NVML\\_SUCCESS](#) if counters were successfully reset
- [NVML\\_ERROR\\_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML\\_ERROR\\_INVALID\\_ARGUMENT](#) if *nvmlIndex* is invalid
- [NVML\\_ERROR\\_NOT\\_SUPPORTED](#) if the device doesn't support this feature
- [NVML\\_ERROR\\_IN\\_USE](#) if the device is still in use and cannot be removed

## 6.17 NvmlClocksThrottleReasons

### Defines

- `#define nvmlClocksThrottleReasonGpuIdle 0x0000000000000001LL`
- `#define nvmlClocksThrottleReasonApplicationsClocksSetting 0x0000000000000002LL`
- `#define nvmlClocksThrottleReasonUserDefinedClocks nvmlClocksThrottleReasonApplicationsClocksSetting`
- `#define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL`
- `#define nvmlClocksThrottleReasonHwSlowdown 0x0000000000000008LL`
- `#define nvmlClocksThrottleReasonSyncBoost 0x0000000000000010LL`
- `#define nvmlClocksThrottleReasonUnknown 0x8000000000000000LL`
- `#define nvmlClocksThrottleReasonNone 0x0000000000000000LL`
- `#define nvmlClocksThrottleReasonAll`

### 6.17.1 Define Documentation

#### 6.17.1.1 `#define nvmlClocksThrottleReasonAll`

##### Value:

```
(nvmlClocksThrottleReasonNone \
| nvmlClocksThrottleReasonGpuIdle \
| nvmlClocksThrottleReasonApplicationsClocksSetting \
| nvmlClocksThrottleReasonSwPowerCap \
| nvmlClocksThrottleReasonHwSlowdown \
| nvmlClocksThrottleReasonSyncBoost \
| nvmlClocksThrottleReasonUnknown \
)
```

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

#### 6.17.1.2 `#define nvmlClocksThrottleReasonApplicationsClocksSetting 0x0000000000000002LL`

GPU clocks are limited by current setting of applications clocks

##### See also:

[nvmlDeviceSetApplicationsClocks](#)  
[nvmlDeviceGetApplicationsClock](#)

#### 6.17.1.3 `#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

**6.17.1.4 #define nvmlClocksThrottleReasonHwSlowdown 0x0000000000000008LL**

HW Slowdown (reducing the core clocks by a factor of 2 or more) is engaged

This is an indicator of:

- temperature being too high
- External Power Brake Assertion is triggered (e.g. by the system power supply)
- Power draw is too high and Fast Trigger protection is reducing the clocks
- May be also reported during PState or clock change
  - This behavior may be removed in a later release.

See also:

[nvmlDeviceGetTemperature](#)  
[nvmlDeviceGetTemperatureThreshold](#)  
[nvmlDeviceGetPowerUsage](#)

**6.17.1.5 #define nvmlClocksThrottleReasonNone 0x0000000000000000LL**

Bit mask representing no clocks throttling

Clocks are as high as possible.

**6.17.1.6 #define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL**

SW Power Scaling algorithm is reducing the clocks below requested clocks

See also:

[nvmlDeviceGetPowerUsage](#)  
[nvmlDeviceSetPowerManagementLimit](#)  
[nvmlDeviceGetPowerManagementLimit](#)

**6.17.1.7 #define nvmlClocksThrottleReasonSyncBoost 0x0000000000000010LL**

Sync Boost

This GPU has been added to a Sync boost group with nvidia-smi or DCGM in order to maximize performance per watt. All GPUs in the sync boost group will boost to the minimum possible clocks across the entire group. Look at the throttle reasons for other GPUs in the system to see why those GPUs are holding this one at lower clocks.

**6.17.1.8 #define nvmlClocksThrottleReasonUnknown 0x8000000000000000LL**

Some other unspecified factor is reducing the clocks

#### 6.17.1.9 `#define nvmlClocksThrottleReasonUserDefinedClocks nvmlClocksThrottleReasonApplicationsClocksSetting`

##### Deprecated

Renamed to [nvmlClocksThrottleReasonApplicationsClocksSetting](#) as the name describes the situation more accurately.

## Chapter 7

# Data Structure Documentation

### 7.1 nvmlAccountingStats\_t Struct Reference

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

#### Data Fields

- unsigned int [gpuUtilization](#)  
*Percent of time over the process's lifetime during which one or more kernels was executing on the GPU. Utilization stats just like returned by [nvmlDeviceGetUtilizationRates](#) but for the life time of a process (not just the last sample period). Set to NVML\_VALUE\_NOT\_AVAILABLE if [nvmlDeviceGetUtilizationRates](#) is not supported.*
- unsigned int [memoryUtilization](#)  
*Percent of time over the process's lifetime during which global (device) memory was being read or written. Set to NVML\_VALUE\_NOT\_AVAILABLE if [nvmlDeviceGetUtilizationRates](#) is not supported.*
- unsigned long long [maxMemoryUsage](#)  
*Maximum total memory in bytes that was ever allocated by the process. Set to NVML\_VALUE\_NOT\_AVAILABLE if [nvmlProcessInfo\\_t->usedGpuMemory](#) is not supported.*
- unsigned long long [time](#)  
*Amount of time in ms during which the compute context was active. The time is reported as 0 if < the process is not terminated.*
- unsigned long long [startTime](#)  
*CPU Timestamp in usec representing start time for the process.*
- unsigned int [isRunning](#)  
*Flag to represent if the process is running (1 for running, 0 for terminated).*
- unsigned int [reserved](#) [5]  
*Reserved for future use.*

### 7.1.1 Detailed Description

Describes accounting statistics of a process.

## 7.2 nvmlBAR1Memory\_t Struct Reference

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

### Data Fields

- unsigned long long [bar1Total](#)  
*Total BAR1 Memory (in bytes).*
- unsigned long long [bar1Free](#)  
*Unallocated BAR1 Memory (in bytes).*
- unsigned long long [bar1Used](#)  
*Allocated Used Memory (in bytes).*

### 7.2.1 Detailed Description

BAR1 Memory allocation Information for a device

## 7.3 nvmlBridgeChipHierarchy\_t Struct Reference

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

### Data Fields

- unsigned char [bridgeCount](#)  
*Number of Bridge Chips on the Board.*
- [nvmlBridgeChipInfo\\_t bridgeChipInfo](#) [NVML\_MAX\_PHYSICAL\_BRIDGE]  
*Hierarchy of Bridge Chips on the board.*

### 7.3.1 Detailed Description

This structure stores the complete Hierarchy of the Bridge Chip within the board. The immediate bridge is stored at index 0 of bridgeInfoList, parent to immediate bridge is at index 1 and so forth.



## 7.4 nvmlBridgeChipInfo\_t Struct Reference

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

### Data Fields

- [nvmlBridgeChipType\\_t](#) type  
*Type of Bridge Chip.*
- unsigned int [fwVersion](#)  
*Firmware Version. 0=Version is unavailable.*

### 7.4.1 Detailed Description

Information about the Bridge Chip Firmware

## 7.5 nvmlEccErrorCounts\_t Struct Reference

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

### Data Fields

- unsigned long long [l1Cache](#)  
*L1 cache errors.*
- unsigned long long [l2Cache](#)  
*L2 cache errors.*
- unsigned long long [deviceMemory](#)  
*Device memory errors.*
- unsigned long long [registerFile](#)  
*Register file errors.*

### 7.5.1 Detailed Description

Detailed ECC error counts for a device.

#### Deprecated

Different GPU families can have different memory error counters See [nvmlDeviceGetMemoryErrorCounter](#)

## 7.6 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.*
- unsigned long long [eventData](#)  
*Stores last XID error for the device in the event of nvmlEventTypeXidCriticalError,.*

### 7.6.1 Detailed Description

Information about occurred event

## 7.7 nvmlHwbcEntry\_t Struct Reference

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

### 7.7.1 Detailed Description

Description of HWBC entry

## 7.8 nvmlLedState\_t Struct Reference

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

### Data Fields

- char [cause](#) [256]  
*If amber, a text description of the cause.*
- [nvmlLedColor\\_t](#) [color](#)  
*GREEN or AMBER.*

### 7.8.1 Detailed Description

LED states for an S-class unit.

## 7.9 nvmlMemory\_t Struct Reference

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

### Data Fields

- unsigned long long [total](#)  
*Total installed FB memory (in bytes).*
- unsigned long long [free](#)  
*Unallocated FB memory (in bytes).*
- unsigned long long [used](#)  
*Allocated FB memory (in bytes). Note that the driver/GPU always sets aside a small amount of memory for bookkeeping.*

### 7.9.1 Detailed Description

Memory allocation information for a device.

## 7.10 nvmlNvLinkUtilizationControl\_t Struct Reference

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

### 7.10.1 Detailed Description

Struct to define the NVLINK counter controls

## 7.11 nvmlPciInfo\_t Struct Reference

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

### Data Fields

- char [busId](#) [NVML\_DEVICE\_PCI\_BUS\_ID\_BUFFER\_SIZE]  
*The tuple domain:bus:device.function PCI identifier (& NULL terminator).*
- unsigned int [domain](#)  
*The PCI domain on which the device's bus resides, 0 to 0xffff.*
- unsigned int [bus](#)  
*The bus on which the device resides, 0 to 0xff.*
- unsigned int [device](#)  
*The device's id on the bus, 0 to 31.*
- unsigned int [pciDeviceId](#)  
*The combined 16-bit device id and 16-bit vendor id.*
- unsigned int [pciSubSystemId](#)  
*The 32-bit Sub System Device ID.*

### 7.11.1 Detailed Description

PCI information about a GPU device.



## 7.12 nvmlProcessInfo\_t Struct Reference

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

### Data Fields

- unsigned int [pid](#)  
*Process ID.*
- unsigned long long [usedGpuMemory](#)  
*Amount of used GPU memory in bytes. Under WDDM, [NVML\\_VALUE\\_NOT\\_AVAILABLE](#) is always reported because Windows KMD manages all the memory and not the NVIDIA driver.*

### 7.12.1 Detailed Description

Information about running compute processes on the GPU

## 7.13 nvmlPSUInfo\_t Struct Reference

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

### Data Fields

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

### 7.13.1 Detailed Description

Power usage information for an S-class unit. The power supply state is a human readable string that equals "Normal" or contains a combination of "Abnormal" plus one or more of the following:

- High voltage
- Fan failure
- Heatsink temperature
- Current limit
- Voltage below UV alarm threshold
- Low-voltage
- SI2C remote off command
- MOD\_DISABLE input
- Short pin transition

## 7.14 nvmlSample\_t Struct Reference

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

### Data Fields

- unsigned long long [timeStamp](#)  
*CPU Timestamp in microseconds.*
- [nvmlValue\\_t](#) [sampleValue](#)  
*Sample Value.*

### 7.14.1 Detailed Description

Information for Sample

## 7.15 nvmlUnitFanInfo\_t Struct Reference

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

### Data Fields

- unsigned int [speed](#)  
*Fan speed (RPM).*
- [nvmlFanState\\_t](#) *state*  
*Flag that indicates whether fan is working properly.*

### 7.15.1 Detailed Description

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

## 7.16 nvmlUnitFanSpeeds\_t Struct Reference

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

### Data Fields

- [nvmlUnitFanInfo\\_t fans](#) [24]  
*Fan speed data for each fan.*
- unsigned int [count](#)  
*Number of fans in unit.*

### 7.16.1 Detailed Description

Fan speed readings for an entire S-class unit.

## 7.17 nvmlUnitInfo\_t Struct Reference

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

### Data Fields

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

### 7.17.1 Detailed Description

Static S-class unit info.

## 7.18 nvmlUtilization\_t Struct Reference

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

### Data Fields

- unsigned int [gpu](#)  
*Percent of time over the past sample period during which one or more kernels was executing on the GPU.*
- unsigned int [memory](#)  
*Percent of time over the past sample period during which global (device) memory was being read or written.*

### 7.18.1 Detailed Description

Utilization information for a device. Each sample period may be between 1 second and 1/6 second, depending on the product being queried.

## 7.19 nvmlValue\_t Union Reference

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

### Data Fields

- double [dVal](#)  
*If the value is double.*
- unsigned int [uiVal](#)  
*If the value is unsigned int.*
- unsigned long [ulVal](#)  
*If the value is unsigned long.*
- unsigned long long [ullVal](#)  
*If the value is unsigned long long.*

### 7.19.1 Detailed Description

Union to represent different types of Value



## 7.20 nvmlViolationTime\_t Struct Reference

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

### Data Fields

- unsigned long long [referenceTime](#)  
*referenceTime represents CPU timestamp in microseconds*
- unsigned long long [violationTime](#)  
*violationTime in Nanoseconds*

### 7.20.1 Detailed Description

Struct to hold perf policy violation status data

# Index

Accounting Statistics, [29](#)

Constants, [36](#)

Device Commands, [90](#)

Device Enums, [18](#)

Device Queries, [43](#)

Device Structs, [15](#)

Drain states, [107](#)

Error reporting, [35](#)

Event Handling Methods, [103](#)

Event Types, [27](#)

Initialization and Cleanup, [33](#)

NvLink

    nvmmlDeviceFreezeNvLinkUtilizationCounter, [97](#)

    nvmmlDeviceGetNvLinkCapability, [98](#)

    nvmmlDeviceGetNvLinkErrorCounter, [98](#)

    nvmmlDeviceGetNvLinkRemotePciInfo, [98](#)

    nvmmlDeviceGetNvLinkState, [99](#)

    nvmmlDeviceGetNvLinkUtilizationControl, [99](#)

    nvmmlDeviceGetNvLinkUtilizationCounter, [100](#)

    nvmmlDeviceGetNvLinkVersion, [100](#)

    nvmmlDeviceResetNvLinkErrorCounters, [101](#)

    nvmmlDeviceResetNvLinkUtilizationCounter, [101](#)

    nvmmlDeviceSetNvLinkUtilizationControl, [101](#)

NvLink Methods, [97](#)

NVML\_AGGREGATE\_ECC

    nvmmlDeviceEnumvs, [22](#)

NVML\_CLOCK\_GRAPHICS

    nvmmlDeviceEnumvs, [21](#)

NVML\_CLOCK\_ID\_APP\_CLOCK\_DEFAULT

    nvmmlDeviceEnumvs, [21](#)

NVML\_CLOCK\_ID\_APP\_CLOCK\_TARGET

    nvmmlDeviceEnumvs, [21](#)

NVML\_CLOCK\_ID\_CURRENT

    nvmmlDeviceEnumvs, [21](#)

NVML\_CLOCK\_ID\_CUSTOMER\_BOOST\_MAX

    nvmmlDeviceEnumvs, [21](#)

NVML\_CLOCK\_MEM

    nvmmlDeviceEnumvs, [21](#)

NVML\_CLOCK\_SM

    nvmmlDeviceEnumvs, [21](#)

NVML\_CLOCK\_VIDEO

    nvmmlDeviceEnumvs, [21](#)

NVML\_COMPUTEMODE\_DEFAULT

    nvmmlDeviceEnumvs, [21](#)

NVML\_COMPUTEMODE\_EXCLUSIVE\_PROCESS

    nvmmlDeviceEnumvs, [21](#)

NVML\_COMPUTEMODE\_EXCLUSIVE\_THREAD

    nvmmlDeviceEnumvs, [21](#)

NVML\_COMPUTEMODE\_PROHIBITED

    nvmmlDeviceEnumvs, [21](#)

NVML\_DEC\_UTILIZATION\_SAMPLES

    nvmmlDeviceStructs, [17](#)

NVML\_DRIVER\_WDDM

    nvmmlDeviceEnumvs, [22](#)

NVML\_DRIVER\_WDM

    nvmmlDeviceEnumvs, [22](#)

NVML\_ECC\_COUNTER\_TYPE\_COUNT

    nvmmlDeviceEnumvs, [22](#)

NVML\_ENC\_UTILIZATION\_SAMPLES

    nvmmlDeviceStructs, [17](#)

NVML\_ERROR\_ALREADY\_INITIALIZED

    nvmmlDeviceEnumvs, [24](#)

NVML\_ERROR\_CORRUPTED\_INFOROM

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_DRIVER\_NOT\_LOADED

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_FUNCTION\_NOT\_FOUND

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_GPU\_IS\_LOST

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_IN\_USE

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_INSUFFICIENT\_POWER

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_INSUFFICIENT\_SIZE

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_INVALID\_ARGUMENT

    nvmmlDeviceEnumvs, [24](#)

NVML\_ERROR\_IRQ\_ISSUE

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_LIB\_RM\_VERSION\_MISMATCH

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_LIBRARY\_NOT\_FOUND

    nvmmlDeviceEnumvs, [25](#)

NVML\_ERROR\_NO\_PERMISSION

    nvmmlDeviceEnumvs, [24](#)

- NVML\_ERROR\_NOT\_FOUND
  - nvmlDeviceEnumvs, [25](#)
- NVML\_ERROR\_NOT\_SUPPORTED
  - nvmlDeviceEnumvs, [24](#)
- NVML\_ERROR\_OPERATING\_SYSTEM
  - nvmlDeviceEnumvs, [25](#)
- NVML\_ERROR\_RESET\_REQUIRED
  - nvmlDeviceEnumvs, [25](#)
- NVML\_ERROR\_TIMEOUT
  - nvmlDeviceEnumvs, [25](#)
- NVML\_ERROR\_UNINITIALIZED
  - nvmlDeviceEnumvs, [24](#)
- NVML\_ERROR\_UNKNOWN
  - nvmlDeviceEnumvs, [25](#)
- NVML\_FAN\_FAILED
  - nvmlUnitStructs, [26](#)
- NVML\_FAN\_NORMAL
  - nvmlUnitStructs, [26](#)
- NVML\_FEATURE\_DISABLED
  - nvmlDeviceEnumvs, [22](#)
- NVML\_FEATURE\_ENABLED
  - nvmlDeviceEnumvs, [22](#)
- NVML\_GOM\_ALL\_ON
  - nvmlDeviceEnumvs, [22](#)
- NVML\_GOM\_COMPUTE
  - nvmlDeviceEnumvs, [22](#)
- NVML\_GOM\_LOW\_DP
  - nvmlDeviceEnumvs, [22](#)
- NVML\_GPU\_UTILIZATION\_SAMPLES
  - nvmlDeviceStructs, [17](#)
- NVML\_INFOROM\_COUNT
  - nvmlDeviceEnumvs, [23](#)
- NVML\_INFOROM\_ECC
  - nvmlDeviceEnumvs, [23](#)
- NVML\_INFOROM\_OEM
  - nvmlDeviceEnumvs, [23](#)
- NVML\_INFOROM\_POWER
  - nvmlDeviceEnumvs, [23](#)
- NVML\_LED\_COLOR\_AMBER
  - nvmlUnitStructs, [26](#)
- NVML\_LED\_COLOR\_GREEN
  - nvmlUnitStructs, [26](#)
- NVML\_MEMORY\_CLK\_SAMPLES
  - nvmlDeviceStructs, [17](#)
- NVML\_MEMORY\_ERROR\_TYPE\_CORRECTED
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_ERROR\_TYPE\_COUNT
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_ERROR\_TYPE\_UNCORRECTED
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_LOCATION\_COUNT
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_LOCATION\_DEVICE\_MEMORY
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_LOCATION\_L1\_CACHE
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_LOCATION\_L2\_CACHE
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_LOCATION\_REGISTER\_FILE
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_LOCATION\_TEXTURE\_-  
MEMORY
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_LOCATION\_TEXTURE\_SHM
  - nvmlDeviceEnumvs, [23](#)
- NVML\_MEMORY\_UTILIZATION\_SAMPLES
  - nvmlDeviceStructs, [17](#)
- NVML\_PAGE\_RETIREMENT\_CAUSE\_DOUBLE\_-  
BIT\_ECC\_ERROR
  - nvmlDeviceEnumvs, [23](#)
- NVML\_PAGE\_RETIREMENT\_CAUSE\_MULTIPLE\_-  
SINGLE\_BIT\_ECC\_ERRORS
  - nvmlDeviceEnumvs, [23](#)
- NVML\_PROCESSOR\_CLK\_SAMPLES
  - nvmlDeviceStructs, [17](#)
- NVML\_PSTATE\_0
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_1
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_10
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_11
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_12
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_13
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_14
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_15
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_2
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_3
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_4
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_5
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_6
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_7
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_8
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_9
  - nvmlDeviceEnumvs, [24](#)
- NVML\_PSTATE\_UNKNOWN

- nvmlDeviceEnumvs, 24
- NVML\_RESTRICTED\_API\_SET\_APPLICATION\_-  
CLOCKS
  - nvmlDeviceEnumvs, 24
- NVML\_RESTRICTED\_API\_SET\_AUTO\_-  
BOOSTED\_CLOCKS
  - nvmlDeviceEnumvs, 24
- NVML\_SUCCESS
  - nvmlDeviceEnumvs, 24
- NVML\_TEMPERATURE\_GPU
  - nvmlDeviceEnumvs, 25
- NVML\_TOTAL\_POWER\_SAMPLES
  - nvmlDeviceStructs, 17
- NVML\_VOLATILE\_ECC
  - nvmlDeviceEnumvs, 22
- NVML\_DEVICE\_INFOROM\_VERSION\_BUFFER\_-  
SIZE
  - nvmlConstants, 36
- NVML\_DEVICE\_NAME\_BUFFER\_SIZE
  - nvmlConstants, 36
- NVML\_DEVICE\_PART\_NUMBER\_BUFFER\_SIZE
  - nvmlConstants, 36
- NVML\_DEVICE\_PCI\_BUS\_ID\_BUFFER\_SIZE
  - nvmlDeviceStructs, 16
- NVML\_DEVICE\_SERIAL\_BUFFER\_SIZE
  - nvmlConstants, 36
- NVML\_DEVICE\_UUID\_BUFFER\_SIZE
  - nvmlConstants, 36
- NVML\_DEVICE\_VBIOS\_VERSION\_BUFFER\_SIZE
  - nvmlConstants, 36
- NVML\_DOUBLE\_BIT\_ECC
  - nvmlDeviceEnumvs, 20
- NVML\_MAX\_PHYSICAL\_BRIDGE
  - nvmlDeviceStructs, 16
- NVML\_NVLINK\_MAX\_LINKS
  - nvmlDeviceStructs, 16
- NVML\_SINGLE\_BIT\_ECC
  - nvmlDeviceEnumvs, 20
- NVML\_SYSTEM\_DRIVER\_VERSION\_BUFFER\_-  
SIZE
  - nvmlConstants, 36
- NVML\_SYSTEM\_NVML\_VERSION\_BUFFER\_SIZE
  - nvmlConstants, 36
- NVML\_VALUE\_NOT\_AVAILABLE
  - nvmlDeviceStructs, 16
- nvmlAccountingStats
  - nvmlDeviceClearAccountingPids, 29
  - nvmlDeviceGetAccountingBufferSize, 29
  - nvmlDeviceGetAccountingMode, 30
  - nvmlDeviceGetAccountingPids, 30
  - nvmlDeviceGetAccountingStats, 31
  - nvmlDeviceSetAccountingMode, 32
- nvmlAccountingStats\_t, 113
- nvmlBAR1Memory\_t, 115
- nvmlBrandType\_t
  - nvmlDeviceEnumvs, 21
- nvmlBridgeChipHierarchy\_t, 116
- nvmlBridgeChipInfo\_t, 117
- nvmlBridgeChipType\_t
  - nvmlDeviceStructs, 16
- nvmlClockId\_t
  - nvmlDeviceEnumvs, 21
- nvmlClocksThrottleReasonAll
  - nvmlClocksThrottleReasons, 110
- nvmlClocksThrottleReasonApplicationsClocksSetting
  - nvmlClocksThrottleReasons, 110
- nvmlClocksThrottleReasonGpuIdle
  - nvmlClocksThrottleReasons, 110
- nvmlClocksThrottleReasonHwSlowdown
  - nvmlClocksThrottleReasons, 110
- nvmlClocksThrottleReasonNone
  - nvmlClocksThrottleReasons, 111
- NvmlClocksThrottleReasons, 110
- nvmlClocksThrottleReasons
  - nvmlClocksThrottleReasonAll, 110
  - nvmlClocksThrottleReasonApplicationsClocksSet-  
ting, 110
  - nvmlClocksThrottleReasonGpuIdle, 110
  - nvmlClocksThrottleReasonHwSlowdown, 110
  - nvmlClocksThrottleReasonNone, 111
  - nvmlClocksThrottleReasonSwPowerCap, 111
  - nvmlClocksThrottleReasonSyncBoost, 111
  - nvmlClocksThrottleReasonUnknown, 111
  - nvmlClocksThrottleReasonUserDefinedClocks, 111
- nvmlClocksThrottleReasonSwPowerCap
  - nvmlClocksThrottleReasons, 111
- nvmlClocksThrottleReasonSyncBoost
  - nvmlClocksThrottleReasons, 111
- nvmlClocksThrottleReasonUnknown
  - nvmlClocksThrottleReasons, 111
- nvmlClocksThrottleReasonUserDefinedClocks
  - nvmlClocksThrottleReasons, 111
- nvmlClockType\_t
  - nvmlDeviceEnumvs, 21
- nvmlComputeMode\_t
  - nvmlDeviceEnumvs, 21
- nvmlConstants
  - NVML\_DEVICE\_INFOROM\_VERSION\_-  
BUFFER\_SIZE, 36
  - NVML\_DEVICE\_NAME\_BUFFER\_SIZE, 36
  - NVML\_DEVICE\_PART\_NUMBER\_BUFFER\_-  
SIZE, 36
  - NVML\_DEVICE\_SERIAL\_BUFFER\_SIZE, 36
  - NVML\_DEVICE\_UUID\_BUFFER\_SIZE, 36
  - NVML\_DEVICE\_VBIOS\_VERSION\_BUFFER\_-  
SIZE, 36
  - NVML\_SYSTEM\_DRIVER\_VERSION\_-  
BUFFER\_SIZE, 36

- NVML\_SYSTEM\_NVML\_VERSION\_BUFFER\_SIZE, 36
- nvmlDeviceClearAccountingPids
  - nvmlAccountingStats, 29
- nvmlDeviceClearCpuAffinity
  - nvmlDeviceQueries, 45
- nvmlDeviceClearEccErrorCounts
  - nvmlDeviceCommands, 90
- nvmlDeviceCommands
  - nvmlDeviceClearEccErrorCounts, 90
  - nvmlDeviceSetAPIRestriction, 91
  - nvmlDeviceSetApplicationsClocks, 91
  - nvmlDeviceSetComputeMode, 92
  - nvmlDeviceSetDriverModel, 93
  - nvmlDeviceSetEccMode, 93
  - nvmlDeviceSetGpuOperationMode, 94
  - nvmlDeviceSetPersistenceMode, 95
  - nvmlDeviceSetPowerManagementLimit, 95
- nvmlDeviceDiscoverGpus
  - nvmlZPI, 107
- nvmlDeviceEnumvs
  - NVML\_AGGREGATE\_ECC, 22
  - NVML\_CLOCK\_GRAPHICS, 21
  - NVML\_CLOCK\_ID\_APP\_CLOCK\_DEFAULT, 21
  - NVML\_CLOCK\_ID\_APP\_CLOCK\_TARGET, 21
  - NVML\_CLOCK\_ID\_CURRENT, 21
  - NVML\_CLOCK\_ID\_CUSTOMER\_BOOST\_MAX, 21
  - NVML\_CLOCK\_MEM, 21
  - NVML\_CLOCK\_SM, 21
  - NVML\_CLOCK\_VIDEO, 21
  - NVML\_COMPUTEMODE\_DEFAULT, 21
  - NVML\_COMPUTEMODE\_EXCLUSIVE\_PROCESS, 21
  - NVML\_COMPUTEMODE\_EXCLUSIVE\_THREAD, 21
  - NVML\_COMPUTEMODE\_PROHIBITED, 21
  - NVML\_DRIVER\_WDDM, 22
  - NVML\_DRIVER\_WDM, 22
  - NVML\_ECC\_COUNTER\_TYPE\_COUNT, 22
  - NVML\_ERROR\_ALREADY\_INITIALIZED, 24
  - NVML\_ERROR\_CORRUPTED\_INFOROM, 25
  - NVML\_ERROR\_DRIVER\_NOT\_LOADED, 25
  - NVML\_ERROR\_FUNCTION\_NOT\_FOUND, 25
  - NVML\_ERROR\_GPU\_IS\_LOST, 25
  - NVML\_ERROR\_IN\_USE, 25
  - NVML\_ERROR\_INSUFFICIENT\_POWER, 25
  - NVML\_ERROR\_INSUFFICIENT\_SIZE, 25
  - NVML\_ERROR\_INVALID\_ARGUMENT, 24
  - NVML\_ERROR\_IRQ\_ISSUE, 25
  - NVML\_ERROR\_LIB\_RM\_VERSION\_MISMATCH, 25
  - NVML\_ERROR\_LIBRARY\_NOT\_FOUND, 25
  - NVML\_ERROR\_NO\_PERMISSION, 24
  - NVML\_ERROR\_NOT\_FOUND, 25
  - NVML\_ERROR\_NOT\_SUPPORTED, 24
  - NVML\_ERROR\_OPERATING\_SYSTEM, 25
  - NVML\_ERROR\_RESET\_REQUIRED, 25
  - NVML\_ERROR\_TIMEOUT, 25
  - NVML\_ERROR\_UNINITIALIZED, 24
  - NVML\_ERROR\_UNKNOWN, 25
  - NVML\_FEATURE\_DISABLED, 22
  - NVML\_FEATURE\_ENABLED, 22
  - NVML\_GOM\_ALL\_ON, 22
  - NVML\_GOM\_COMPUTE, 22
  - NVML\_GOM\_LOW\_DP, 22
  - NVML\_INFOROM\_COUNT, 23
  - NVML\_INFOROM\_ECC, 23
  - NVML\_INFOROM\_OEM, 23
  - NVML\_INFOROM\_POWER, 23
  - NVML\_MEMORY\_ERROR\_TYPE\_CORRECTED, 23
  - NVML\_MEMORY\_ERROR\_TYPE\_COUNT, 23
  - NVML\_MEMORY\_ERROR\_TYPE\_UNCORRECTED, 23
  - NVML\_MEMORY\_LOCATION\_COUNT, 23
  - NVML\_MEMORY\_LOCATION\_DEVICE\_MEMORY, 23
  - NVML\_MEMORY\_LOCATION\_L1\_CACHE, 23
  - NVML\_MEMORY\_LOCATION\_L2\_CACHE, 23
  - NVML\_MEMORY\_LOCATION\_REGISTER\_FILE, 23
  - NVML\_MEMORY\_LOCATION\_TEXTURE\_MEMORY, 23
  - NVML\_MEMORY\_LOCATION\_TEXTURE\_SHM, 23
  - NVML\_PAGE\_RETIREMENT\_CAUSE\_DOUBLE\_BIT\_ECC\_ERROR, 23
  - NVML\_PAGE\_RETIREMENT\_CAUSE\_MULTIPLE\_SINGLE\_BIT\_ECC\_ERRORS, 23
  - NVML\_PSTATE\_0, 24
  - NVML\_PSTATE\_1, 24
  - NVML\_PSTATE\_10, 24
  - NVML\_PSTATE\_11, 24
  - NVML\_PSTATE\_12, 24
  - NVML\_PSTATE\_13, 24
  - NVML\_PSTATE\_14, 24
  - NVML\_PSTATE\_15, 24
  - NVML\_PSTATE\_2, 24
  - NVML\_PSTATE\_3, 24
  - NVML\_PSTATE\_4, 24
  - NVML\_PSTATE\_5, 24
  - NVML\_PSTATE\_6, 24
  - NVML\_PSTATE\_7, 24
  - NVML\_PSTATE\_8, 24
  - NVML\_PSTATE\_9, 24
  - NVML\_PSTATE\_UNKNOWN, 24

- NVML\_RESTRICTED\_API\_SET\_-  
APPLICATION\_CLOCKS, 24
- NVML\_RESTRICTED\_API\_SET\_AUTO\_-  
BOOSTED\_CLOCKS, 24
- NVML\_SUCCESS, 24
- NVML\_TEMPERATURE\_GPU, 25
- NVML\_VOLATILE\_ECC, 22
- NVML\_DOUBLE\_BIT\_ECC, 20
- NVML\_SINGLE\_BIT\_ECC, 20
- nvmlBrandType\_t, 21
- nvmlClockId\_t, 21
- nvmlClockType\_t, 21
- nvmlComputeMode\_t, 21
- nvmlDriverModel\_t, 21
- nvmlEccBitType\_t, 20
- nvmlEccCounterType\_t, 22
- nvmlEnableState\_t, 22
- nvmlGpuOperationMode\_t, 22
- nvmlInforomObject\_t, 22
- nvmlMemoryErrorType\_t, 23
- nvmlMemoryLocation\_t, 23
- nvmlPageRetirementCause\_t, 23
- nvmlPstates\_t, 23
- nvmlRestrictedAPI\_t, 24
- nvmlReturn\_t, 24
- nvmlTemperatureSensors\_t, 25
- nvmlTemperatureThresholds\_t, 25
- nvmlDeviceFreezeNvLinkUtilizationCounter  
NvLink, 97
- nvmlDeviceGetAccountingBufferSize  
nvmlAccountingStats, 29
- nvmlDeviceGetAccountingMode  
nvmlAccountingStats, 30
- nvmlDeviceGetAccountingPids  
nvmlAccountingStats, 30
- nvmlDeviceGetAccountingStats  
nvmlAccountingStats, 31
- nvmlDeviceGetAPIRestriction  
nvmlDeviceQueries, 46
- nvmlDeviceGetApplicationsClock  
nvmlDeviceQueries, 46
- nvmlDeviceGetAutoBoostedClocksEnabled  
nvmlDeviceQueries, 47
- nvmlDeviceGetBAR1MemoryInfo  
nvmlDeviceQueries, 47
- nvmlDeviceGetBoardId  
nvmlDeviceQueries, 48
- nvmlDeviceGetBoardPartNumber  
nvmlDeviceQueries, 48
- nvmlDeviceGetBrand  
nvmlDeviceQueries, 49
- nvmlDeviceGetBridgeChipInfo  
nvmlDeviceQueries, 49
- nvmlDeviceGetClock  
nvmlDeviceQueries, 49
- nvmlDeviceGetClockInfo  
nvmlDeviceQueries, 50
- nvmlDeviceGetComputeMode  
nvmlDeviceQueries, 50
- nvmlDeviceGetComputeRunningProcesses  
nvmlDeviceQueries, 51
- nvmlDeviceGetCount  
nvmlDeviceQueries, 52
- nvmlDeviceGetCpuAffinity  
nvmlDeviceQueries, 52
- nvmlDeviceGetCurrentClocksThrottleReasons  
nvmlDeviceQueries, 53
- nvmlDeviceGetCurrPcieLinkGeneration  
nvmlDeviceQueries, 53
- nvmlDeviceGetCurrPcieLinkWidth  
nvmlDeviceQueries, 54
- nvmlDeviceGetDecoderUtilization  
nvmlDeviceQueries, 54
- nvmlDeviceGetDefaultApplicationsClock  
nvmlDeviceQueries, 54
- nvmlDeviceGetDetailedEccErrors  
nvmlDeviceQueries, 55
- nvmlDeviceGetDisplayActive  
nvmlDeviceQueries, 56
- nvmlDeviceGetDisplayMode  
nvmlDeviceQueries, 56
- nvmlDeviceGetDriverModel  
nvmlDeviceQueries, 57
- nvmlDeviceGetEccMode  
nvmlDeviceQueries, 57
- nvmlDeviceGetEncoderUtilization  
nvmlDeviceQueries, 58
- nvmlDeviceGetEnforcedPowerLimit  
nvmlDeviceQueries, 58
- nvmlDeviceGetFanSpeed  
nvmlDeviceQueries, 59
- nvmlDeviceGetGpuOperationMode  
nvmlDeviceQueries, 59
- nvmlDeviceGetGraphicsRunningProcesses  
nvmlDeviceQueries, 60
- nvmlDeviceGetHandleByIndex  
nvmlDeviceQueries, 61
- nvmlDeviceGetHandleByPciBusId  
nvmlDeviceQueries, 62
- nvmlDeviceGetHandleBySerial  
nvmlDeviceQueries, 62
- nvmlDeviceGetHandleByUUID  
nvmlDeviceQueries, 63
- nvmlDeviceGetIndex  
nvmlDeviceQueries, 64
- nvmlDeviceGetInforomConfigurationChecksum  
nvmlDeviceQueries, 64
- nvmlDeviceGetInforomImageVersion

- [nvmlDeviceQueries](#), 65
- [nvmlDeviceGetInforomVersion](#)
  - [nvmlDeviceQueries](#), 65
- [nvmlDeviceGetMaxClockInfo](#)
  - [nvmlDeviceQueries](#), 66
- [nvmlDeviceGetMaxCustomerBoostClock](#)
  - [nvmlDeviceQueries](#), 67
- [nvmlDeviceGetMaxPcieLinkGeneration](#)
  - [nvmlDeviceQueries](#), 67
- [nvmlDeviceGetMaxPcieLinkWidth](#)
  - [nvmlDeviceQueries](#), 68
- [nvmlDeviceGetMemoryErrorCounter](#)
  - [nvmlDeviceQueries](#), 68
- [nvmlDeviceGetMemoryInfo](#)
  - [nvmlDeviceQueries](#), 69
- [nvmlDeviceGetMinorNumber](#)
  - [nvmlDeviceQueries](#), 69
- [nvmlDeviceGetMultiGpuBoard](#)
  - [nvmlDeviceQueries](#), 70
- [nvmlDeviceGetName](#)
  - [nvmlDeviceQueries](#), 70
- [nvmlDeviceGetNvLinkCapability](#)
  - [NvLink](#), 98
- [nvmlDeviceGetNvLinkErrorCounter](#)
  - [NvLink](#), 98
- [nvmlDeviceGetNvLinkRemotePciInfo](#)
  - [NvLink](#), 98
- [nvmlDeviceGetNvLinkState](#)
  - [NvLink](#), 99
- [nvmlDeviceGetNvLinkUtilizationControl](#)
  - [NvLink](#), 99
- [nvmlDeviceGetNvLinkUtilizationCounter](#)
  - [NvLink](#), 100
- [nvmlDeviceGetNvLinkVersion](#)
  - [NvLink](#), 100
- [nvmlDeviceGetPcieReplayCounter](#)
  - [nvmlDeviceQueries](#), 71
- [nvmlDeviceGetPcieThroughput](#)
  - [nvmlDeviceQueries](#), 71
- [nvmlDeviceGetPciInfo](#)
  - [nvmlDeviceQueries](#), 72
- [nvmlDeviceGetPerformanceState](#)
  - [nvmlDeviceQueries](#), 72
- [nvmlDeviceGetPersistenceMode](#)
  - [nvmlDeviceQueries](#), 72
- [nvmlDeviceGetPowerManagementDefaultLimit](#)
  - [nvmlDeviceQueries](#), 73
- [nvmlDeviceGetPowerManagementLimit](#)
  - [nvmlDeviceQueries](#), 73
- [nvmlDeviceGetPowerManagementLimitConstraints](#)
  - [nvmlDeviceQueries](#), 74
- [nvmlDeviceGetPowerManagementMode](#)
  - [nvmlDeviceQueries](#), 74
- [nvmlDeviceGetPowerState](#)
  - [nvmlDeviceQueries](#), 75
- [nvmlDeviceGetPowerUsage](#)
  - [nvmlDeviceQueries](#), 76
- [nvmlDeviceGetRetiredPages](#)
  - [nvmlDeviceQueries](#), 76
- [nvmlDeviceGetRetiredPagesPendingStatus](#)
  - [nvmlDeviceQueries](#), 77
- [nvmlDeviceGetSamples](#)
  - [nvmlDeviceQueries](#), 77
- [nvmlDeviceGetSerial](#)
  - [nvmlDeviceQueries](#), 78
- [nvmlDeviceGetSupportedClocksThrottleReasons](#)
  - [nvmlDeviceQueries](#), 78
- [nvmlDeviceGetSupportedEventTypes](#)
  - [nvmlEvents](#), 103
- [nvmlDeviceGetSupportedGraphicsClocks](#)
  - [nvmlDeviceQueries](#), 79
- [nvmlDeviceGetSupportedMemoryClocks](#)
  - [nvmlDeviceQueries](#), 80
- [nvmlDeviceGetTemperature](#)
  - [nvmlDeviceQueries](#), 80
- [nvmlDeviceGetTemperatureThreshold](#)
  - [nvmlDeviceQueries](#), 81
- [nvmlDeviceGetTopologyCommonAncestor](#)
  - [nvmlDeviceQueries](#), 81
- [nvmlDeviceGetTopologyNearestGpus](#)
  - [nvmlDeviceQueries](#), 81
- [nvmlDeviceGetTotalEccErrors](#)
  - [nvmlDeviceQueries](#), 82
- [nvmlDeviceGetUtilizationRates](#)
  - [nvmlDeviceQueries](#), 83
- [nvmlDeviceGetUUID](#)
  - [nvmlDeviceQueries](#), 83
- [nvmlDeviceGetVbiosVersion](#)
  - [nvmlDeviceQueries](#), 84
- [nvmlDeviceGetViolationStatus](#)
  - [nvmlDeviceQueries](#), 84
- [nvmlDeviceModifyDrainState](#)
  - [nvmlZPI](#), 107
- [nvmlDeviceOnSameBoard](#)
  - [nvmlDeviceQueries](#), 85
- [nvmlDeviceQueries](#)
  - [nvmlDeviceClearCpuAffinity](#), 45
  - [nvmlDeviceGetAPIRestriction](#), 46
  - [nvmlDeviceGetApplicationsClock](#), 46
  - [nvmlDeviceGetAutoBoostedClocksEnabled](#), 47
  - [nvmlDeviceGetBAR1MemoryInfo](#), 47
  - [nvmlDeviceGetBoardId](#), 48
  - [nvmlDeviceGetBoardPartNumber](#), 48
  - [nvmlDeviceGetBrand](#), 49
  - [nvmlDeviceGetBridgeChipInfo](#), 49
  - [nvmlDeviceGetClock](#), 49
  - [nvmlDeviceGetClockInfo](#), 50
  - [nvmlDeviceGetComputeMode](#), 50



- `nvmlDeviceGetComputeRunningProcesses`, 51
- `nvmlDeviceGetCount`, 52
- `nvmlDeviceGetCpuAffinity`, 52
- `nvmlDeviceGetCurrentClocksThrottleReasons`, 53
- `nvmlDeviceGetCurrPcieLinkGeneration`, 53
- `nvmlDeviceGetCurrPcieLinkWidth`, 54
- `nvmlDeviceGetDecoderUtilization`, 54
- `nvmlDeviceGetDefaultApplicationsClock`, 54
- `nvmlDeviceGetDetailedEccErrors`, 55
- `nvmlDeviceGetDisplayActive`, 56
- `nvmlDeviceGetDisplayMode`, 56
- `nvmlDeviceGetDriverModel`, 57
- `nvmlDeviceGetEccMode`, 57
- `nvmlDeviceGetEncoderUtilization`, 58
- `nvmlDeviceGetEnforcedPowerLimit`, 58
- `nvmlDeviceGetFanSpeed`, 59
- `nvmlDeviceGetGpuOperationMode`, 59
- `nvmlDeviceGetGraphicsRunningProcesses`, 60
- `nvmlDeviceGetHandleByIndex`, 61
- `nvmlDeviceGetHandleByPciBusId`, 62
- `nvmlDeviceGetHandleBySerial`, 62
- `nvmlDeviceGetHandleByUUID`, 63
- `nvmlDeviceGetIndex`, 64
- `nvmlDeviceGetInforomConfigurationChecksum`, 64
- `nvmlDeviceGetInforomImageVersion`, 65
- `nvmlDeviceGetInforomVersion`, 65
- `nvmlDeviceGetMaxClockInfo`, 66
- `nvmlDeviceGetMaxCustomerBoostClock`, 67
- `nvmlDeviceGetMaxPcieLinkGeneration`, 67
- `nvmlDeviceGetMaxPcieLinkWidth`, 68
- `nvmlDeviceGetMemoryErrorCounter`, 68
- `nvmlDeviceGetMemoryInfo`, 69
- `nvmlDeviceGetMinorNumber`, 69
- `nvmlDeviceGetMultiGpuBoard`, 70
- `nvmlDeviceGetName`, 70
- `nvmlDeviceGetPcieReplayCounter`, 71
- `nvmlDeviceGetPcieThroughput`, 71
- `nvmlDeviceGetPciInfo`, 72
- `nvmlDeviceGetPerformanceState`, 72
- `nvmlDeviceGetPersistenceMode`, 72
- `nvmlDeviceGetPowerManagementDefaultLimit`, 73
- `nvmlDeviceGetPowerManagementLimit`, 73
- `nvmlDeviceGetPowerManagementLimitConstraints`, 74
- `nvmlDeviceGetPowerManagementMode`, 74
- `nvmlDeviceGetPowerState`, 75
- `nvmlDeviceGetPowerUsage`, 76
- `nvmlDeviceGetRetiredPages`, 76
- `nvmlDeviceGetRetiredPagesPendingStatus`, 77
- `nvmlDeviceGetSamples`, 77
- `nvmlDeviceGetSerial`, 78
- `nvmlDeviceGetSupportedClocksThrottleReasons`, 78
- `nvmlDeviceGetSupportedGraphicsClocks`, 79
- `nvmlDeviceGetSupportedMemoryClocks`, 80
- `nvmlDeviceGetTemperature`, 80
- `nvmlDeviceGetTemperatureThreshold`, 81
- `nvmlDeviceGetTopologyCommonAncestor`, 81
- `nvmlDeviceGetTopologyNearestGpus`, 81
- `nvmlDeviceGetTotalEccErrors`, 82
- `nvmlDeviceGetUtilizationRates`, 83
- `nvmlDeviceGetUUID`, 83
- `nvmlDeviceGetVbiosVersion`, 84
- `nvmlDeviceGetViolationStatus`, 84
- `nvmlDeviceOnSameBoard`, 85
- `nvmlDeviceResetApplicationsClocks`, 85
- `nvmlDeviceSetAutoBoostedClocksEnabled`, 86
- `nvmlDeviceSetCpuAffinity`, 86
- `nvmlDeviceSetDefaultAutoBoostedClocksEnabled`, 87
- `nvmlDeviceValidateInforom`, 87
- `nvmlSystemGetTopologyGpuSet`, 88
- `nvmlDeviceQueryDrainState`
  - `nvmlZPI`, 108
- `nvmlDeviceRegisterEvents`
  - `nvmlEvents`, 104
- `nvmlDeviceRemoveGpu`
  - `nvmlZPI`, 108
- `nvmlDeviceResetApplicationsClocks`
  - `nvmlDeviceQueries`, 85
- `nvmlDeviceResetNvLinkErrorCounters`
  - `NvLink`, 101
- `nvmlDeviceResetNvLinkUtilizationCounter`
  - `NvLink`, 101
- `nvmlDeviceSetAccountingMode`
  - `nvmlAccountingStats`, 32
- `nvmlDeviceSetAPIRestriction`
  - `nvmlDeviceCommands`, 91
- `nvmlDeviceSetApplicationsClocks`
  - `nvmlDeviceCommands`, 91
- `nvmlDeviceSetAutoBoostedClocksEnabled`
  - `nvmlDeviceQueries`, 86
- `nvmlDeviceSetComputeMode`
  - `nvmlDeviceCommands`, 92
- `nvmlDeviceSetCpuAffinity`
  - `nvmlDeviceQueries`, 86
- `nvmlDeviceSetDefaultAutoBoostedClocksEnabled`
  - `nvmlDeviceQueries`, 87
- `nvmlDeviceSetDriverModel`
  - `nvmlDeviceCommands`, 93
- `nvmlDeviceSetEccMode`
  - `nvmlDeviceCommands`, 93
- `nvmlDeviceSetGpuOperationMode`
  - `nvmlDeviceCommands`, 94
- `nvmlDeviceSetNvLinkUtilizationControl`
  - `NvLink`, 101
- `nvmlDeviceSetPersistenceMode`
  - `nvmlDeviceCommands`, 95



- nvmlDeviceSetPowerManagementLimit
  - nvmlDeviceCommands, 95
- nvmlDeviceStructs
  - NVML\_DEC\_UTILIZATION\_SAMPLES, 17
  - NVML\_ENC\_UTILIZATION\_SAMPLES, 17
  - NVML\_GPU\_UTILIZATION\_SAMPLES, 17
  - NVML\_MEMORY\_CLK\_SAMPLES, 17
  - NVML\_MEMORY\_UTILIZATION\_SAMPLES, 17
  - NVML\_PROCESSOR\_CLK\_SAMPLES, 17
  - NVML\_TOTAL\_POWER\_SAMPLES, 17
  - NVML\_DEVICE\_PCI\_BUS\_ID\_BUFFER\_SIZE, 16
  - NVML\_MAX\_PHYSICAL\_BRIDGE, 16
  - NVML\_NVLINK\_MAX\_LINKS, 16
  - NVML\_VALUE\_NOT\_AVAILABLE, 16
  - nvmlBridgeChipType\_t, 16
  - nvmlGpuTopologyLevel\_t, 16
  - nvmlNvLinkCapability\_t, 16
  - nvmlNvLinkErrorCounter\_t, 16
  - nvmlNvLinkUtilizationCountPktTypes\_t, 17
  - nvmlNvLinkUtilizationCountUnits\_t, 17
  - nvmlPcieUtilCounter\_t, 17
  - nvmlPerfPolicyType\_t, 17
  - nvmlSamplingType\_t, 17
  - nvmlValueType\_t, 17
- nvmlDeviceValidateInforom
  - nvmlDeviceQueries, 87
- nvmlDriverModel\_t
  - nvmlDeviceEnumvs, 21
- nvmlEccBitType\_t
  - nvmlDeviceEnumvs, 20
- nvmlEccCounterType\_t
  - nvmlDeviceEnumvs, 22
- nvmlEccErrorCounts\_t, 118
- nvmlEnableState\_t
  - nvmlDeviceEnumvs, 22
- nvmlErrorReporting
  - nvmlErrorString, 35
- nvmlErrorString
  - nvmlErrorReporting, 35
- nvmlEventData\_t, 119
- nvmlEvents
  - nvmlDeviceGetSupportedEventTypes, 103
  - nvmlDeviceRegisterEvents, 104
  - nvmlEventSet\_t, 103
  - nvmlEventSetCreate, 104
  - nvmlEventSetFree, 105
  - nvmlEventSetWait, 105
- nvmlEventSet\_t
  - nvmlEvents, 103
- nvmlEventSetCreate
  - nvmlEvents, 104
- nvmlEventSetFree
  - nvmlEvents, 105
- nvmlEventSetWait
  - nvmlEvents, 105
- nvmlEventType
  - nvmlEventTypeClock, 27
  - nvmlEventTypeDoubleBitEccError, 27
  - nvmlEventTypePState, 27
  - nvmlEventTypeSingleBitEccError, 28
- nvmlEventTypeClock
  - nvmlEventType, 27
- nvmlEventTypeDoubleBitEccError
  - nvmlEventType, 27
- nvmlEventTypePState
  - nvmlEventType, 27
- nvmlEventTypeSingleBitEccError
  - nvmlEventType, 28
- nvmlFanState\_t
  - nvmlUnitStructs, 26
- nvmlGpuOperationMode\_t
  - nvmlDeviceEnumvs, 22
- nvmlGpuTopologyLevel\_t
  - nvmlDeviceStructs, 16
- nvmlHwbcEntry\_t, 120
- nvmlInforomObject\_t
  - nvmlDeviceEnumvs, 22
- nvmlInit
  - nvmlInitializationAndCleanup, 33
- nvmlInitializationAndCleanup
  - nvmlInit, 33
  - nvmlShutdown, 33
- nvmlLedColor\_t
  - nvmlUnitStructs, 26
- nvmlLedState\_t, 121
- nvmlMemory\_t, 122
- nvmlMemoryErrorType\_t
  - nvmlDeviceEnumvs, 23
- nvmlMemoryLocation\_t
  - nvmlDeviceEnumvs, 23
- nvmlNvLinkCapability\_t
  - nvmlDeviceStructs, 16
- nvmlNvLinkErrorCounter\_t
  - nvmlDeviceStructs, 16
- nvmlNvLinkUtilizationControl\_t, 123
- nvmlNvLinkUtilizationCountPktTypes\_t
  - nvmlDeviceStructs, 17
- nvmlNvLinkUtilizationCountUnits\_t
  - nvmlDeviceStructs, 17
- nvmlPageRetirementCause\_t
  - nvmlDeviceEnumvs, 23
- nvmlPcieUtilCounter\_t
  - nvmlDeviceStructs, 17
- nvmlPciInfo\_t, 124
- nvmlPerfPolicyType\_t
  - nvmlDeviceStructs, 17

- nvmlProcessInfo\_t, 125
- nvmlPstates\_t
  - nvmlDeviceEnumvs, 23
- nvmlPSUInfo\_t, 126
- nvmlRestrictedAPI\_t
  - nvmlDeviceEnumvs, 24
- nvmlReturn\_t
  - nvmlDeviceEnumvs, 24
- nvmlSample\_t, 127
- nvmlSamplingType\_t
  - nvmlDeviceStructs, 17
- nvmlShutdown
  - nvmlInitializationAndCleanup, 33
- nvmlSystemGetDriverVersion
  - nvmlSystemQueries, 37
- nvmlSystemGetHicVersion
  - nvmlUnitQueries, 39
- nvmlSystemGetNVMLVersion
  - nvmlSystemQueries, 37
- nvmlSystemGetProcessName
  - nvmlSystemQueries, 38
- nvmlSystemGetTopologyGpuSet
  - nvmlDeviceQueries, 88
- nvmlSystemQueries
  - nvmlSystemGetDriverVersion, 37
  - nvmlSystemGetNVMLVersion, 37
  - nvmlSystemGetProcessName, 38
- nvmlTemperatureSensors\_t
  - nvmlDeviceEnumvs, 25
- nvmlTemperatureThresholds\_t
  - nvmlDeviceEnumvs, 25
- nvmlUnitCommands
  - nvmlUnitSetLedState, 89
- nvmlUnitFanInfo\_t, 128
- nvmlUnitFanSpeeds\_t, 129
- nvmlUnitGetCount
  - nvmlUnitQueries, 39
- nvmlUnitGetDevices
  - nvmlUnitQueries, 40
- nvmlUnitGetFanSpeedInfo
  - nvmlUnitQueries, 40
- nvmlUnitGetHandleByIndex
  - nvmlUnitQueries, 40
- nvmlUnitGetLedState
  - nvmlUnitQueries, 41
- nvmlUnitGetPsuInfo
  - nvmlUnitQueries, 41
- nvmlUnitGetTemperature
  - nvmlUnitQueries, 42
- nvmlUnitGetUnitInfo
  - nvmlUnitQueries, 42
- nvmlUnitInfo\_t, 130
- nvmlUnitQueries
  - nvmlSystemGetHicVersion, 39
  - nvmlUnitGetCount, 39
  - nvmlUnitGetDevices, 40
  - nvmlUnitGetFanSpeedInfo, 40
  - nvmlUnitGetHandleByIndex, 40
  - nvmlUnitGetLedState, 41
  - nvmlUnitGetPsuInfo, 41
  - nvmlUnitGetTemperature, 42
  - nvmlUnitGetUnitInfo, 42
- nvmlUnitSetLedState
  - nvmlUnitCommands, 89
- nvmlUnitStructs
  - NVML\_FAN\_FAILED, 26
  - NVML\_FAN\_NORMAL, 26
  - NVML\_LED\_COLOR\_AMBER, 26
  - NVML\_LED\_COLOR\_GREEN, 26
  - nvmlFanState\_t, 26
  - nvmlLedColor\_t, 26
- nvmlUtilization\_t, 131
- nvmlValue\_t, 132
- nvmlValueType\_t
  - nvmlDeviceStructs, 17
- nvmlViolationTime\_t, 133
- nvmlZPI
  - nvmlDeviceDiscoverGpus, 107
  - nvmlDeviceModifyDrainState, 107
  - nvmlDeviceQueryDrainState, 108
  - nvmlDeviceRemoveGpu, 108
- System Queries, 37
- Unit Commands, 89
- Unit Queries, 39
- Unit Structs, 26

### **Notice**

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

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

### **Trademarks**

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

### **Copyright**

© 2007-2016 NVIDIA Corporation. All rights reserved.