

NVML API Reference Guide

Release 620

NVIDIA Corporation

Nov 11, 2026

Contents

1 NVML API Reference **3**
1.1 API Documentation . . . 3

2 Known Issues **5**

3 Change Log **7**
3.1 Changes between NVML v615 and v620 . . . 7
3.2 Changes between NVML v610 and v615 . . . 8
3.3 Changes between NVML v595 and v610 . . . 9
3.4 Changes between NVML v590 and v595 . . . 10
3.5 Changes between v580 and v590 . . . 12
3.6 Changes between v575 and v580 . . . 13
3.7 Changes between v570 Update and v575 . . . 14
3.8 Changes between v565 and v570 . . . 15
3.9 Changes between v560 and v565 . . . 17
3.10 Changes between v555 and v560 . . . 18
3.11 Changes between v550 and v555 . . . 18
3.12 Changes between v545 and v550 . . . 18
3.13 Changes between v535 and v545 . . . 19
3.14 Changes between v530 and v535 . . . 20
3.15 Changes between v525 and v530 . . . 21
3.16 Changes between v520 and v525 . . . 21
3.17 Changes between v515 and v520 . . . 22
3.18 Changes between v510 and v515 . . . 23
3.19 Changes between v495 and v510 . . . 23
3.20 Changes between v465 and v470 . . . 23
3.21 Changes between v460 and v465 . . . 24
3.22 Changes between v450 and v460 . . . 24
3.23 Changes between v445 and v450 . . . 24
3.24 Changes between v418 and v445 . . . 24
3.25 Changes between v361 and v418 . . . 25
3.26 Changes between v349 and v361 . . . 25
3.27 Changes between v346 and v349 . . . 25
3.28 Changes between v340 and v346 . . . 25
3.29 Changes between NVML v331 and v340 . . . 26
3.30 Changes between NVML v5.319 Update and v331 . . . 26
3.31 Changes between NVML v5.319 RC and v5.319 Update . . . 27
3.32 Changes between NVML v4.304 Production and v5.319 RC . . . 27
3.33 Changes between NVML v4.304 RC and v4.304 Production . . . 27
3.34 Changes between NVML v3.295 and v4.304 RC . . . 28
3.35 Changes between NVML v2.285 and v3.295 . . . 28
3.36 Changes between NVML v1.0 and v2.285 . . . 29

4 Deprecation and/or Removal Notices for the NVML Library **31**
4.1 CUDA 13.0 . . . 31

5 NVML API Reference **33**
5.1 Namespace hierarchy . . . 33
5.2 API contents . . . 38
5.2.1 Groups . . . 38
5.2.1.1 Accounting Statistics . . . 38
5.2.1.2 CPER (Common Platform Error Record) . . . 45
5.2.1.3 Confidential Computing definitions . . . 46
5.2.1.4 Constants . . . 52
5.2.1.5 Device Commands . . . 53
5.2.1.6 Device Enums . . . 70
5.2.1.7 Device Queries . . . 95
5.2.1.8 Device Structs . . . 215
5.2.1.9 Drain State definitions . . . 234
5.2.1.10 Drain states . . . 234
5.2.1.11 Encoder Structs . . . 237
5.2.1.12 Error reporting . . . 238
5.2.1.13 Event Handling Methods . . . 238
5.2.1.14 Excluded GPU Queries . . . 260
5.2.1.15 Fabric definitions . . . 261
5.2.1.16 Field Value Enums . . . 269
5.2.1.17 Field Value Queries . . . 308
5.2.1.18 Frame Buffer Capture Structures . . . 309
5.2.1.19 Initialization and Cleanup . . . 311
5.2.1.20 MPS Locality Domain Device Enumeration . . . 314
5.2.1.21 Multi Instance GPU Management . . . 314
5.2.1.22 NVML GPM . . . 339
5.2.1.23 NvLink Methods . . . 374
5.2.1.24 NvmlClocksEventReasons . . . 394
5.2.1.25 PRM Access . . . 399
5.2.1.26 Power Profile Information . . . 403
5.2.1.27 Power Smoothing Information . . . 410
5.2.1.28 System Queries . . . 414
5.2.1.29 Unit Commands . . . 420
5.2.1.30 Unit Queries . . . 421
5.2.1.31 Unit Structs . . . 425
5.2.1.32 vGPU APIs . . . 426
5.2.1.33 vGPU Enums, Constants, Structs . . . 435
5.2.1.34 vGPU Management . . . 450
5.2.1.35 vGPU Migration . . . 480
5.2.1.36 vGPU Utilization and Accounting . . . 488
5.2.2 Pages . . . 499
5.2.2.1 Deprecated List . . . 499
5.2.3 Structs . . . 502
5.2.3.1 nvmlAccountingStats_t . . . 502
5.2.3.2 nvmlAccountingStats_v2_t . . . 503
5.2.3.3 nvmlActiveVgpuInstanceInfo_v1_t . . . 504
5.2.3.4 nvmlAdaptiveTgpModeInfo_v1_t . . . 504
5.2.3.5 nvmlBAR1Memory_t . . . 505
5.2.3.6 nvmlBBXTimeData_v1_t . . . 505
5.2.3.7 nvmlBridgeChipHierarchy_t . . . 506
5.2.3.8 nvmlBridgeChipInfo_t . . . 506

5.2.3.9 nvmlC2cModeInfo_v1_t . . . 506
5.2.3.10 nvmlCPERCursor_v1_t . . . 506
5.2.3.11 nvmlClkMonFaultInfo_t . . . 507
5.2.3.12 nvmlClkMonStatus_t . . . 507
5.2.3.13 nvmlClockOffset_v1_t . . . 508
5.2.3.14 nvmlComputeInstanceInfo_t . . . 508
5.2.3.15 nvmlComputeInstancePlacement_t . . . 509
5.2.3.16 nvmlComputeInstanceProfileInfo_t . . . 509
5.2.3.17 nvmlComputeInstanceProfileInfo_v2_t . . . 510
5.2.3.18 nvmlComputeInstanceProfileInfo_v3_t . . . 511
5.2.3.19 nvmlConfComputeGetKeyRotationThresholdInfo_v1_t . . . 512
5.2.3.20 nvmlConfComputeGpuAttestationReport_t . . . 512
5.2.3.21 nvmlConfComputeGpuCertificate_t . . . 512
5.2.3.22 nvmlConfComputeMemSizeInfo_t . . . 513
5.2.3.23 nvmlConfComputeSetKeyRotationThresholdInfo_v1_t . . . 513
5.2.3.24 nvmlConfComputeSystemCaps_t . . . 513
5.2.3.25 nvmlConfComputeSystemState_t . . . 513
5.2.3.26 nvmlCoolerInfo_v1_t . . . 514
5.2.3.27 nvmlCoreRailMetrics_t . . . 514
5.2.3.28 nvmlDeviceAddressingMode_v1_t . . . 514
5.2.3.29 nvmlDeviceAttributes_t . . . 515
5.2.3.30 nvmlDeviceCapabilities_v1_t . . . 515
5.2.3.31 nvmlDeviceCurrentClockFreqs_v1_t . . . 516
5.2.3.32 nvmlDevicePerfModes_v1_t . . . 516
5.2.3.33 nvmlDevicePowerMizerModes_v1_t . . . 516
5.2.3.34 nvmlDramEncryptionInfo_v1_t . . . 517
5.2.3.35 nvmlEccBankRemapperHistogram_v1_t . . . 517
5.2.3.36 nvmlEccBankRemapperStatus_v1_t . . . 517
5.2.3.37 nvmlEccErrorCounts_t . . . 518
5.2.3.38 nvmlEccSramErrorStatus_v1_t . . . 518
5.2.3.39 nvmlEccSramUniqueUncorrectedErrorCounts_v1_t . . . 519
5.2.3.40 nvmlEccSramUniqueUncorrectedErrorEntry_v1_t . . . 520
5.2.3.41 nvmlEncoderSessionInfo_t . . . 520
5.2.3.42 nvmlEventData_t . . . 521
5.2.3.43 nvmlEventSetGetContextCount_v1_t . . . 522
5.2.3.44 nvmlEventSetGetContextData_v1_t . . . 522
5.2.3.45 nvmlEventSetGetContextInfo_v1_t . . . 522
5.2.3.46 nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t . . . 523
5.2.3.47 nvmlEventSetWait_v3_t . . . 523
5.2.3.48 nvmlExcludedDeviceInfo_t . . . 525
5.2.3.49 nvmlFBCSessionInfo_t . . . 525
5.2.3.50 nvmlFBCStats_t . . . 526
5.2.3.51 nvmlFanSpeedInfo_v1_t . . . 527
5.2.3.52 nvmlFieldValue_t . . . 527
5.2.3.53 nvmlGetCPER_v1_t . . . 528
5.2.3.54 nvmlGetMemoryLimits_v1_t . . . 528
5.2.3.55 nvmlGpmMetric_t . . . 529
5.2.3.56 nvmlGpmMetricsGet_t . . . 529
5.2.3.57 nvmlGpmSupport_t . . . 530
5.2.3.58 nvmlGpuDynamicPstatesInfo_t . . . 530
5.2.3.59 nvmlGpuFabricClique_v1_t . . . 531
5.2.3.60 nvmlGpuFabricInfo_t . . . 531
5.2.3.61 nvmlGpuFabricInfo_v2_t . . . 532
5.2.3.62 nvmlGpuFabricInfo_v3_t . . . 532

5.2.3.63 nvmlGpuFabricInfo_v4_t . . . 533
5.2.3.64 nvmlGpuInstanceInfo_t . . . 534
5.2.3.65 nvmlGpuInstancePlacement_t . . . 534
5.2.3.66 nvmlGpuInstanceProfileInfo_t . . . 534
5.2.3.67 nvmlGpuInstanceProfileInfo_v2_t . . . 535
5.2.3.68 nvmlGpuInstanceProfileInfo_v3_t . . . 536
5.2.3.69 nvmlGpuOperationalEventConfig_v1_t . . . 537
5.2.3.70 nvmlGpuThermalSettings_t . . . 538
5.2.3.71 nvmlGpuUptime_v1_t . . . 538
5.2.3.72 nvmlGridLicensableFeature_t . . . 539
5.2.3.73 nvmlGridLicensableFeatures_t . . . 539
5.2.3.74 nvmlGridLicenseExpiry_t . . . 540
5.2.3.75 nvmlHostname_v1_t . . . 540
5.2.3.76 nvmlHwbcEntry_t . . . 540
5.2.3.77 nvmlLedState_t . . . 541
5.2.3.78 nvmlMarginTemperature_v1_t . . . 541
5.2.3.79 nvmlMask255_t . . . 541
5.2.3.80 nvmlMemory_t . . . 541
5.2.3.81 nvmlMemory_v2_t . . . 542
5.2.3.82 nvmlNvLinkInfo_v1_t . . . 542
5.2.3.83 nvmlNvLinkInfo_v2_t . . . 543
5.2.3.84 nvmlNvLinkInfo_v3_t . . . 543
5.2.3.85 nvmlNvLinkPowerThres_t . . . 543
5.2.3.86 nvmlNvLinkUtilizationControl_t . . . 544
5.2.3.87 nvmlNvlinkFirmwareInfo_t . . . 544
5.2.3.88 nvmlNvlinkFirmwareVersion_t . . . 544
5.2.3.89 nvmlNvlinkGetBwMode_v1_t . . . 544
5.2.3.90 nvmlNvlinkGetRemapTableInfo_v1_t . . . 545
5.2.3.91 nvmlNvlinkLockRemapTable_v1_t . . . 545
5.2.3.92 nvmlNvlinkNvlePrivateInfo_v1_t . . . 545
5.2.3.93 nvmlNvlinkSetBwModeAsync_v1_t . . . 546
5.2.3.94 nvmlNvlinkSetBwMode_v1_t . . . 546
5.2.3.95 nvmlNvlinkSetNvleReady_v1_t . . . 546
5.2.3.96 nvmlNvlinkSetupNvleEncryptionKey_v1_t . . . 547
5.2.3.97 nvmlNvlinkSupportedBwModes_v1_t . . . 547
5.2.3.98 nvmlNvlinkTelemetrySample_v1_t . . . 547
5.2.3.99 nvmlNvlinkTelemetrySamples_v1_t . . . 548
5.2.3.100 nvmlObservedMetrics_t . . . 548
5.2.3.101 nvmlPRMCounterInput_v1_t . . . 549
5.2.3.102 nvmlPRMCounterList_v1_t . . . 549
5.2.3.103 nvmlPRMCounterValue_v1_t . . . 549
5.2.3.104 nvmlPRMCounter_v1_t . . . 550
5.2.3.105 nvmlPRMTLV_v1_t . . . 550
5.2.3.106 nvmlPSUInfo_t . . . 551
5.2.3.107 nvmlPciInfoExt_v1_t . . . 551
5.2.3.108 nvmlPciInfo_t . . . 552
5.2.3.109 nvmlPdi_v1_t . . . 553
5.2.3.110 nvmlPerfMetricControllerSample_t . . . 553
5.2.3.111 nvmlPerfMetricsDlppc2xSample_t . . . 553
5.2.3.112 nvmlPerfMetricsPfpp1xSample_t . . . 554
5.2.3.113 nvmlPerfMetricsSample_t . . . 554
5.2.3.114 nvmlPerfMetricsSamples_v1_t . . . 554
5.2.3.115 nvmlPlatformInfo_v1_t . . . 555
5.2.3.116 nvmlPlatformInfo_v2_t . . . 555

5.2.3.117 nvmlPmgrPwrTuple_t . . . 556
5.2.3.118 nvmlPowerSmoothingProfile_v1_t . . . 556
5.2.3.119 nvmlPowerSmoothingState_v1_t . . . 557
5.2.3.120 nvmlPowerValue_v2_t . . . 557
5.2.3.121 nvmlProcessDetailList_v1_t . . . 558
5.2.3.122 nvmlProcessDetail_v1_t . . . 558
5.2.3.123 nvmlProcessInfo_t . . . 558
5.2.3.124 nvmlProcessInfo_v1_t . . . 559
5.2.3.125 nvmlProcessUtilizationInfo_v1_t . . . 559
5.2.3.126 nvmlProcessUtilizationSample_t . . . 560
5.2.3.127 nvmlProcessesUtilizationInfo_v1_t . . . 560
5.2.3.128 nvmlPwrModelMetricsDlppm1xDramclkEstimates_t . . . 561
5.2.3.129 nvmlPwrModelMetricsDlppm1xPerf_t . . . 561
5.2.3.130 nvmlPwrModelMetricsDlppm1x_t . . . 561
5.2.3.131 nvmlPwrModelMetricsPfpp1x_t . . . 562
5.2.3.132 nvmlPwrModelMetricsSamplePfpp1x_t . . . 563
5.2.3.133 nvmlPwrModelOperatingPointPfpp1x_t . . . 563
5.2.3.134 nvmlRailMetrics_t . . . 563
5.2.3.135 nvmlRemappedRowsInfo_v2_t . . . 563
5.2.3.136 nvmlRepairStatus_v1_t . . . 564
5.2.3.137 nvmlRowRemapperHistogramValues_t . . . 564
5.2.3.138 nvmlRusdSettings_v1_t . . . 565
5.2.3.139 nvmlSample_t . . . 565
5.2.3.140 nvmlSetMemoryLimits_v1_t . . . 565
5.2.3.141 nvmlSystemConfComputeSettings_v1_t . . . 566
5.2.3.142 nvmlSystemConfComputeSettings_v2_t . . . 566
5.2.3.143 nvmlSystemDriverBranchInfo_v1_t . . . 567
5.2.3.144 nvmlSystemEventData_v1_t . . . 567
5.2.3.145 nvmlSystemEventSetCreateRequest_v1_t . . . 567
5.2.3.146 nvmlSystemEventSetFreeRequest_v1_t . . . 568
5.2.3.147 nvmlSystemEventSetWaitRequest_v1_t . . . 568
5.2.3.148 nvmlSystemRegisterEventRequest_v1_t . . . 569
5.2.3.149 nvmlTemperature_v1_t . . . 569
5.2.3.150 nvmlUUID_v1_t . . . 569
5.2.3.151 nvmlUnitFanInfo_t . . . 570
5.2.3.152 nvmlUnitFanSpeeds_t . . . 570
5.2.3.153 nvmlUnitInfo_t . . . 570
5.2.3.154 nvmlUnrepairableMemoryStatus_v1_t . . . 571
5.2.3.155 nvmlUtilization_t . . . 571
5.2.3.156 nvmlVgpuCreatablePlacementInfo_v1_t . . . 571
5.2.3.157 nvmlVgpuHeterogeneousMode_v1_t . . . 572
5.2.3.158 nvmlVgpuInstanceUtilizationInfo_v1_t . . . 572
5.2.3.159 nvmlVgpuInstanceUtilizationSample_t . . . 573
5.2.3.160 nvmlVgpuInstancesUtilizationInfo_v1_t . . . 573
5.2.3.161 nvmlVgpuLicenseExpiry_t . . . 574
5.2.3.162 nvmlVgpuLicenseInfo_t . . . 574
5.2.3.163 nvmlVgpuMetadata_t . . . 575
5.2.3.164 nvmlVgpuPgpuCompatibility_t . . . 575
5.2.3.165 nvmlVgpuPgpuMetadata_t . . . 576
5.2.3.166 nvmlVgpuPlacementId_v1_t . . . 576
5.2.3.167 nvmlVgpuPlacementList_v1_t . . . 577
5.2.3.168 nvmlVgpuPlacementList_v2_t . . . 577
5.2.3.169 nvmlVgpuProcessUtilizationInfo_v1_t . . . 578
5.2.3.170 nvmlVgpuProcessUtilizationSample_t . . . 579

5.2.3.171 nvmlVgpuProcessesUtilizationInfo_v1_t . . . 579
5.2.3.172 nvmlVgpuRuntimeState_v1_t . . . 580
5.2.3.173 nvmlVgpuSchedulerCapabilities_t . . . 580
5.2.3.174 nvmlVgpuSchedulerGetState_t . . . 581
5.2.3.175 nvmlVgpuSchedulerLogEntry_t . . . 581
5.2.3.176 nvmlVgpuSchedulerLogEntry_v2_t . . . 582
5.2.3.177 nvmlVgpuSchedulerLogInfo_v1_t . . . 582
5.2.3.178 nvmlVgpuSchedulerLogInfo_v2_t . . . 583
5.2.3.179 nvmlVgpuSchedulerLog_t . . . 583
5.2.3.180 nvmlVgpuSchedulerSetState_t . . . 584
5.2.3.181 nvmlVgpuSchedulerStateInfo_v1_t . . . 584
5.2.3.182 nvmlVgpuSchedulerStateInfo_v2_t . . . 585
5.2.3.183 nvmlVgpuSchedulerState_v1_t . . . 585
5.2.3.184 nvmlVgpuSchedulerState_v2_t . . . 586
5.2.3.185 nvmlVgpuTypeBar1Info_v1_t . . . 586
5.2.3.186 nvmlVgpuTypeIdInfo_v1_t . . . 587
5.2.3.187 nvmlVgpuTypeMaxInstance_v1_t . . . 587
5.2.3.188 nvmlVgpuVersion_t . . . 587
5.2.3.189 nvmlViolationTime_t . . . 588
5.2.3.190 nvmlWorkloadPowerProfileCurrentProfiles_v1_t . . . 588
5.2.3.191 nvmlWorkloadPowerProfileInfo_v1_t . . . 588
5.2.3.192 nvmlWorkloadPowerProfileProfilesInfo_v1_t . . . 589
5.2.3.193 nvmlWorkloadPowerProfileRequestedProfiles_v1_t . . . 589
5.2.3.194 nvmlWorkloadPowerProfileUpdateProfiles_v1_t . . . 589
5.2.4 Unions . . . 599
5.2.4.1 nvmlUUIDValue_t . . . 599
5.2.4.2 nvmlValue_t . . . 599
5.2.4.3 nvmlVgpuSchedulerParams_t . . . 600
5.2.4.4 nvmlVgpuSchedulerSetParams_t . . . 600

6 Notices **603**
6.1 Trademarks . . . 604

Index **605**

The NVIDIA Management Library (NVML) is a C-based API for monitoring and managing NVIDIA GPUs. Applications can use NVML to query GPU state, utilization, memory, temperature, power, clocks, topology, processes, and events, and to perform supported management operations.

NVML is intended for developers building GPU monitoring, diagnostics, resource management, and administration tools. It is thread-safe and provides the underlying functionality for the NVIDIA System Management Interface (`nvidia-smi`). Use this guide to find platform and library information, API functions and data structures, known issues, release changes, and deprecation notices.

Chapter 1. NVML API Reference

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

1.1. API Documentation

Supported operating systems and products:

For the list of supported operating systems and Data Center GPU products corresponding to your driver version, refer to the NVIDIA Data Center Documentation. NVML also has limited support for newer non-data-center GPUs (for example, NVIDIA® GeForce® or NVIDIA® RTX® GPUs).

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

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

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

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

The NVML API is divided into five categories:

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

- `Event Handling Methods`
- Error Reporting Methods:
  - `Error Reporting`

Chapter 2. Known Issues

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

- On Vera Rubin systems, *nvmlDeviceGetEnforcedPowerLimit()* and *nvmlDeviceGetFieldValues()* with *NVML_FI_DEV_POWER_CURRENT_LIMIT* may take longer than in previous releases. This will be fixed in a future release.
- NVML Field Values from #251 - #273 (Power Smoothing, Clock Event Reason, and Sync Power Balancing related field values) have changed between 13.0 and 13.0U1/v580TRD2.
  - Any application that is using these field IDs must be recompiled using the NVML header file from CUDA 13.0 Update 1 in order to continue working correctly with NVIDIA drivers v580 TRD2 and beyond.
- On systems where GPUs are NUMA nodes, the accuracy of FB memory utilization provided by NVML depends on the memory accounting of the operating system. This is because FB memory is managed by the operating system instead of the NVIDIA GPU driver. Typically, pages allocated from FB memory are not released even after the process terminates to enhance performance. In scenarios where the operating system is under memory pressure, it may resort to utilizing FB memory. Such actions can result in discrepancies in the accuracy of memory reporting.
- On Linux, GPU Reset can't be triggered when there is a pending GPU Operation Mode (GOM) change.
- On Linux, GPU Reset may not successfully change a pending ECC mode. A full reboot may be required to enable the mode change.
- `nvmlAccountingStats` supports only one process per GPU at a time (CUDA proxy server counts as one process).
- `nvmlAccountingStats_t.time` reports time and utilization values starting from `cuInit` until process termination. Future driver versions might change this behavior slightly and account for the process only from `cuCtxCreate` until `cuCtxDestroy`.
- On GPUs from the Fermi family, current P0 clocks (reported by `nvmlDeviceGetClockInfo`) can differ from max clocks by a few MHz.
- The GPU hostname APIs are currently only supported on compatible GB200 platforms.

Chapter 3. Change Log

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

3.1. Changes between NVML v615 and v620

- Updated the description of *NVML_ERROR_OPERATING_SYSTEM* to clarify that it represents a generic operating system error.
- Added the Linux-only, opt-in MPS locality-domain device enumeration view. Each MPS locality domain has an NVML device handle and index.
  - The restricted view is enabled by setting the environment variable `NVML_MPS_DEVICE_ENUMERATION` to 1 and exposes only its documented API subset. Other public APIs return *NVML_ERROR_NOT_SUPPORTED*.
  - Multiple locality domains can share a physical parent's PCI identity. PCI bus ID lookup returns the first matching handle in NVML enumeration order; use the UUID or index to select a specific locality domain.
  - Loss or invalidation of the selected MPS telemetry session is reported as *NVML_ERROR_INVALID_STATE* without falling back to physical GPU enumeration.
- Added *nvmlDeviceGetNvLinkInfo_v3()* to support NVLink Encryption Ready information.
- Added field ID *NVML_FI_DEV_GPU_TEMP* to report the current GPU temperature in degrees C with value type *NVML_VALUE_TYPE_DOUBLE*.
- Added field ID *NVML_FI_PWR_SMOOTHING_LIMITED_TIME* to report the running total of time, in nanoseconds, that the GPU has been limited by power smoothing, with value type *NVML_VALUE_TYPE_UNSIGNED_LONG_LONG*.
- Deprecated the *NVML_POWER_SMOOTHING_IDX_FROM_FIELD_VAL* macro. It will be removed in a future release.
- Added Baremetal Secure AI feature support.
- Added ARM CCA, Intel TDX Connect, and AMD SEV-TIO CPU capability reporting for Confidential Compute system capabilities:
  - *NVML_CC_SYSTEM_CPU_CAPS_ARM_CCA*
  - *NVML_CC_SYSTEM_CPU_CAPS_INTEL_TDX_CONNECT*
  - *NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV_TIO*
- Added *nvmlSystemConfComputeSettings_v2_t*, which extends *nvmlSystemGetConfComputeSettings()* to report the TDISP state of the GPUs in a single `tdispState` field:

- *NVML_CC_SYSTEM_TDISP_STATE_DISABLED* when the GPUs are not behind a TDI.
- *NVML_CC_SYSTEM_TDISP_STATE_NOT_READY* while the TDIs are in FW-only mode (UNLOCKED).
- *NVML_CC_SYSTEM_TDISP_STATE_READY* once the TDIs have completed the TSM handshake (RUN).

- Added new NVLink Encryption (NVLE) APIs:
  - *nvmlDeviceLockNvlinkRemapTable_v1()* to lock the NVLink remap table and prevent MSE from making further updates.
  - *nvmlDeviceGetNvlinkRemapTableInfo_v1()* to query the NVLink remap table information.
  - *nvmlDeviceGetNvlinkNvlePrivateInfo_v1()* to retrieve opaque NVLE private information for a device.
  - *nvmlDeviceSetupNvlinkNvleEncryptionKey_v1()* to set up the NVLink NVLE encryption key between GPUs.
  - *nvmlDeviceSetNvlinkNvleReady_v1()* to set the NVLink NVLE ready state.
- Added *nvmlDeviceGetGpuUptime_v1()* to query the GPU uptime since initialization by the driver.
- Added vGPU capability *NVML_DEVICE_VGPU_CAP_UVM_STUN* to query whether the GPU supports UVM stun.

3.2. Changes between NVML v610 and v615

- Added *NVML_GPU_RECOVERY_ACTION_BUS_RESET* and *NVML_GPU_RECOVERY_ACTION_SYSTEM_REBOOT* to *nvmlDeviceGpuRecoveryAction_t*.
- Added GPU Operational Event APIs to register and wait for structured events and retrieve their associated context records:
  - *nvmlEventSetRegisterGpuOperationalEvents_v1()* and *nvmlEventSetWait_v3()*
  - *nvmlEventSetGetContextCount_v1()*, *nvmlEventSetGetContextInfo_v1()*, *nvmlEventSetGetContextData_v1()*, and *nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1()*
- Added *nvmlDeviceGetBankRemapperStatus_v1()* to query the bank remapper status.
- Added a new temperature sensor type *NVML_TEMPERATURE_GPU_MAX* to report the temperature from the hottest part of the GPU.
- Added 2 new clock event reason bitmask values for board limit and reliability policies:
  - *nvmlClocksEventReasonBoardLimit*
  - *nvmlClocksEventReasonReliability*
- Added new *NVML_NVLINK_STATE_ACTIVE_TRAFFIC_DISABLED* state.
- Added *nvmlDeviceSetNvlinkBwModeAsync_v1()* to asynchronously set Nvlink Bandwidth mode.
- Added *nvmlDeviceSetMemoryLimits_v1()* to set device memory limits in a cgroup partition.
- Added *nvmlDeviceGetMemoryLimits_v1()* to get device memory limits in a cgroup partition.

- Modified *nvmlDeviceGetMemoryInfo_v2()* to account for cgroup memory partitions when called within the partition.
- Added *nvmlDeviceSetAdaptiveTgpMode_v1()* and *nvmlDeviceGetAdaptiveTgpMode-Info_v1()* to set and query Adaptive TGP mode.
- Added *nvmlDeviceGetNvLinkTelemetrySamples_v1()* to get historical NVLink telemetry samples.
- Modified *nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1()* to allow non-privileged access using WPPS capability.
- Added new Workload Power Profile types to *nvmlPowerProfileType_t*.
- Added GPM metrics `NVML_GPM_METRIC_NVLINK_L[36..71]`.
- Added missing comments on GPM metrics of C2C, cache, NVENC, GR CTXSW, and NVLink PER_SEC.
- Added new GPU fabric clique types and structures for per-type clique information:
  - *NVML_GPU_FABRIC_CLIQUE_MAX*
  - *NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_POINTER*,*NVML_GPU_FABRIC_CLIQUE_TYPE_MULTICAST_POI*
    NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_LOGICAL_ENDPOINT,
    NVML_GPU_FABRIC_CLIQUE_TYPE_MULTICAST_LOGICAL_ENDPOINT
  - *nvmlGpuFabricClique_v1_t*
- Added *nvmlDeviceGetGpuFabricInfo_v4()* to retrieve per-type fabric clique assignments.
- Deprecated *nvmlGpuFabricInfo_v3_t* in favor of *nvmlGpuFabricInfo_v4_t*.
- Added new `NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE*` Health Mask flags to GPU Fabric Health Mask.

3.3. Changes between NVML v595 and v610

- Added new MCLK field IDs:
  - *NVML_FI_DEV_MCLK_SWITCH_TYPE*
  - *NVML_FI_DEV_MCLK_MIN_SWITCH_INTERVAL_MILLISECONDS*
- Added new NVLINK field IDs:
  - *NVML_FI_DEV_NVLINK_PLR_XMIT_BLOCKS*
  - *NVML_FI_DEV_NVLINK_PLR_XMIT_RETRY_BLOCKS*
- Added *nvmlDeviceGetAccountingStats_v2()* to retrieve additional utilization information:
  - `sampleCount`
  - `sumGpuUtil`
  - `sumFbUtil`
- Added a new NVLINK field ID:
  - *NVML_FI_DEV_NVLINK_GET_DATA_RATE*
- Added support for new power scope *NVML_POWER_SCOPE_GPU_BASE*, which represents GPU base power.

- Added support in Power Management APIs:
  - *nvmlDeviceSetPowerManagementLimit_v2()*
- Added support for the following *nvmlFieldValue_t* fields with the new scope:
  - *NVML_FI_DEV_POWER_MIN_LIMIT*
  - *NVML_FI_DEV_POWER_MAX_LIMIT*
  - *NVML_FI_DEV_POWER_DEFAULT_LIMIT*
  - *NVML_FI_DEV_POWER_CURRENT_LIMIT*
  - *NVML_FI_DEV_POWER_REQUESTED_LIMIT*
- Added support for detecting *nvmlClocksEventReasonSwThermalSlowdown* on Thor Jetson platforms through:
  - *nvmlDeviceGetSupportedClocksEventReasons()*
  - *nvmlDeviceGetCurrentClocksEventReasons()*
- Added *NVML_FI_DEV_MMA_STALL_PERCENT* for MMA stall telemetry.
- Updated field ID *NVML_FI_DEV_MMA_STALL_PERCENT* value type from *NVML_VALUE_TYPE_UNSIGNED_INT* to *NVML_VALUE_TYPE_DOUBLE*.
- Added *nvmlDevicePerfMetricsGetSamples_v1()* to get Performance Metric samples.
- Added *NVML_FI_PWR_SMOOTHING_SOC_POWER_SMOOTHING_ENABLED* for State-Of-Charge Power Smoothing support.
- Added public API support for Common Platform Error Record (CPER) data:
  - *nvmlSystemGetCPER_v1()* to retrieve CPER records using a caller-supplied buffer and cursor-based iteration.
  - *nvmlGetCPER_v1_t* for input and output values, including cursor, buffer, and buffer size.
  - *nvmlCPERCursor_v1_t* for query filters and read position, including `cperTypeMask`, `uuid`, and `handle`.
  - *nvmlCPERType_t* and *NVML_CPER_ACCESS_TYPE_GPU* for record type filtering.
  - *NVML_CPER_CURSOR_HANDLE_INIT* for cursor initialization.
  - Added size-query support by calling with `buffer` set to `NULL` and `bufferSize` set to `0`. The API returns *NVML_ERROR_INSUFFICIENT_SIZE* with the required size, or *NVML_SUCCESS* with `bufferSize` set to `0` when no records exist.

3.4. Changes between NVML v590 and v595

- Added new NVLINK field IDs:
  - *NVML_FI_DEV_NVLINK_COUNT_RAW_ERRORS_LANE0*
  - *NVML_FI_DEV_NVLINK_COUNT_RAW_ERRORS_LANE1*
  - *NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE0_V2*
  - *NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE1_V2*
  - *NVML_FI_DEV_NVLINK_COUNT_RAW_BER_V2*

- Updated the descriptions of *nvmlGpmSampleGet()* and *nvmlGpmQueryDeviceSupport()* to emphasize that both device and MIG handles are supported by these APIs.
- Added *NVML_INIT_FLAG_FORCE_INIT* to force GPU initialization when a previous *nvmlInit* was called with NO_GPUS and NO_ATTACH flags.
- Updated the GPM metrics' name from NVLINK1_L[0..17] to NVLINK_L[18..35].
- Updated Nvlink Version to include *NVML_NVLINK_VERSION_6_0*.
- Added a new API to retrieve GPU PRM counters: *nvmlDeviceReadPRMCounters_v1()*.
- Added new *NVML_GPU_RECOVERY_ACTION_RECOVER_IMEX_DOMAIN* enum to nvmlDeviceGpuRecoveryAction_s.
- Added *nvmlDeviceGetBBXTimeData_v1()*, which returns the cumulative number of seconds the GPU has had the driver loaded.
- Added new GPM UTIL metric *NVML_GPM_METRIC_DMMA_TENSOR_UTIL*.
- Added new GPM raw metrics.
- Added *nvmlDeviceGetVgpuSchedulerState_v2()* to fetch the vGPU software scheduler state without taking ARR mode into account.
- Added *nvmlGpuInstanceGetVgpuSchedulerState_v2()* to fetch the vGPU software scheduler state for the given GPU Instance without taking ARR mode into account.
- Added *nvmlDeviceGetVgpuSchedulerLog_v2()* to query new field weight in the vGPU software scheduler logs for the device.
- Added *nvmlGpuInstanceGetVgpuSchedulerLog_v2()* to query new field weight in the vGPU software scheduler logs for the given GPU Instance.
- Added *nvmlDeviceSetVgpuSchedulerState_v2()* to set the vGPU scheduler state, with support for disabling ARR mode removed in this version.
- Added *nvmlGpuInstanceSetVgpuSchedulerState_v2()* to set the vGPU scheduler state for the given GPU Instance, with support for disabling ARR mode removed in this version.
- Deprecated vGPU scheduler APIs:
  - *nvmlDeviceGetVgpuSchedulerState()*
  - *nvmlGpuInstanceGetVgpuSchedulerState()*
  - *nvmlDeviceGetVgpuSchedulerLog()*
  - *nvmlGpuInstanceGetVgpuSchedulerLog()*
  - *nvmlDeviceSetVgpuSchedulerState()*
  - *nvmlGpuInstanceSetVgpuSchedulerState()*
- Added *nvmlDeviceVgpuForceGspUnload()* to force GSP unload on a vGPU host.
- Added new Incorrect Configuration Status to *nvmlGpuFabricInfoV_t*:
  - *NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_GPU_STATE_INVALID*
- Added new NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED* Health Mask flags to GPU Fabric Health Mask.
- Added vGPU 'NVIDIA RTX Virtual Game Dev' as a licensed product for vGPU.
- Added new MIG profile *NVML_GPU_INSTANCE_PROFILE_3_SLICE_GFX* for 3-slice graphics and media support.

3.5. Changes between v580 and v590

The following new functionality is exposed on NVIDIA display drivers version 590 Production or later.

- Added new product architecture *NVML_DEVICE_ARCH_NPU3*.
- Deprecated *NVML_BRAND_NVIDIA_DLA* product brand.
- Added new *NVML_BRAND_NVIDIA_NPU* product brand.
- Added *nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1()* to clear, set, or clobber existing workload power profiles from a single API.
- Deprecated workload power profile APIs, which will be removed in CUDA 15.0:
  - *nvmlDeviceWorkloadPowerProfileSetRequestedProfiles()*
  - *nvmlDeviceWorkloadPowerProfileClearRequestedProfiles()*
- Added *NVML_FI_DEV_EDPP_MULTIPLIER* field value to report the current EDPp multiplier as a percentage.
- Added *nvmlDeviceGetUnrepairableMemoryFlag_v1()* to report whether or not there is unrepairable memory.
- Updated *nvmlDeviceGetVgpuSchedulerLog()* and *nvmlGpuInstanceGetVgpuSchedulerState()* to access the vGPU scheduler logs for the NVENC0 and NVENC1 engines.
- Updated *NVML_NVLINK_MAX_LINKS* to 36 to accommodate for more NvLinks in Rubin.
- Added NVLINK1 GPM metrics for Rubin.
- Added 14 new Power Smoothing field IDs for Delayed Power Smoothing:
  - *NVML_FI_PWR_SMOOTHING_PRIMARY_POWER_FLOOR*
  - *NVML_FI_PWR_SMOOTHING_SECONDARY_POWER_FLOOR*
  - *NVML_FI_PWR_SMOOTHING_MIN_PRIMARY_FLOOR_ACT_OFFSET*
  - *NVML_FI_PWR_SMOOTHING_MIN_PRIMARY_FLOOR_ACT_POINT*
  - *NVML_FI_PWR_SMOOTHING_WINDOW_MULTIPLIER*
  - *NVML_FI_PWR_SMOOTHING_DELAYED_PWR_SMOOTHING_SUPPORTED*
  - *NVML_FI_PWR_SMOOTHING_PROFILE_SECONDARY_POWER_FLOOR*
  - *NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_ACT_WIN_MULT*
  - *NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_TAR_WIN_MULT*
  - *NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_ACT_OFFSET*
  - *NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_SECONDARY_POWER_FLOOR*
  - *NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_ACT_WIN_MULT*
  - *NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_TAR_WIN_MULT*
  - *NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_ACT_OFFSET*
- Added four new Power Smoothing profile parameter IDs to set new Delayed Power Smoothing profile fields:
  - *NVML_POWER_SMOOTHING_PROFILE_PARAM_SECONDARY_POWER_FLOOR*
  - *NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_ACT_WIN_MULT*

- *NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_TAR_WIN_MULT*
- *NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_ACT_OFFSET*

3.6. Changes between v575 and v580

The following new functionality is exposed on NVIDIA display drivers version 580 Production or later.

- Added *nvmlDeviceGetRemappedRows_v2()* to query the number of inactive remapped rows.
- Added *NVML_FI_DEV_REMAPPED_ROWS_COR_INACTIVE* and *NVML_FI_DEV_REMAPPED_ROWS_UNC_INACTIVE* for *nvmlDeviceGetFieldValues()* to query the number of inactive remapped rows.
- Fixed bug with `NVML_FI_PWR_SMOOTHING_*` field value numbering, which was different than the v570 values.
- Adjusted `NVML_FI_DEV_CLOCKS_EVENT_REASON_*` and `NVML_FI_DEV_POWER_SYNC_BALANCING_*` field value numbering to resolve overlap with `NVML_FI_PWR_SMOOTHING_*` field values.
- Added *nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1()* to clear, set, or clobber existing profiles from a single API.
- Added *nvmlDeviceSetRusdSettings_v1()* to set RUSD (read only user shared data buffer) polling mask.
- Added *nvmlDeviceGetSramUniqueUncorrectedEccErrorCounts()* to get the counts of SRAM unique uncorrected ECC errors.
- Deprecated Applications Clocks APIs, which will be removed in CUDA 14.0:
  - *nvmlDeviceSetApplicationsClocks()*
  - *nvmlDeviceGetApplicationsClock()*
  - *nvmlDeviceGetDefaultApplicationsClock()*
  - *nvmlDeviceResetApplicationsClocks()*
- Deprecated *nvmlDeviceGetViolationStatus()*, which will be removed in CUDA 14.0.
- Added *nvmlDeviceGetNvLinkInfo()* to query device NVLINK info.
- Added *nvmlDeviceGetPdi()* to retrieve the device GPU PDI.
- Added Multi-GPU mode NVLINK Encryption *NVML_CC_SYSTEM_MULTIGPU_NVLE*.
- Added V2 struct to *nvmlDeviceGetNvLinkInfo()* to query NVLINK Firmware info.
- Added *nvmlDeviceReadWritePRM_v1()* to retrieve GPU PRM register contents.
- Added *nvmlDeviceGetAddressingMode()* to retrieve the addressing mode for the device.
- Added *nvmlDeviceGetRepairStatus()* to get ECC status info.
- Added DLA device support.
- Added *nvmlDeviceGetGpuInstanceProfileInfoByIdV()*, which allows for MIG GPU instance profile info to be queried with profileId instead of profile name.
- Updated *nvmlGpuFabricInfoV_t* to v3 to include a new Health Summary field and new Incorrect Configuration statuses.
  - *nvmlGpuFabricInfo_v2_t* is deprecated and will be removed in a future release.
  - New Incorrect Configuration Statuses:

    - *NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INCOMPATIBLE_GPU_FW*
    - *NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INVALID_LOCATION*
- Added *nvmlDeviceGetPowerMizerMode_v1()* to query the current and supported power mizer modes on Maxwell and newer GPUs. Power mizer mode provides a hint to the driver as to how to manage GPU performance.
- Added *nvmlDeviceSetPowerMizerMode_v1()* to set the power mizer mode on Maxwell and newer GPUs.
- Added *nvmlDeviceSetHostname_v1()* and *nvmlDeviceGetHostname_v1()* to allow custom GPU hostname configuration.

3.7. Changes between v570 Update and v575

The following new functionality is exposed on NVIDIA display drivers version 575 Production or later.

- Added *nvmlSystemEventSetCreate()* to create a system event set.
- Added *nvmlSystemEventSetFree()* to release a system event set.
- Added *nvmlSystemRegisterEvents()* to register system events on a system event set.
- Added *nvmlSystemEventSetWait()* to wait for system event notification and obtain system event data.
- Added *nvmlGpuInstanceGetCreatableVgpus()* to query the currently creatable vGPU types on the user-provided GPU Instance.
- Added *nvmlVgpuTypeGetMaxInstancesPerGpuInstance()* to query the maximum number of vGPU instances per GPU Instance for the given vGPU type.
- Added *nvmlGpuInstanceSetVgpuSchedulerState()* to set the vGPU scheduler state for the given GPU Instance.
- Added *nvmlGpuInstanceGetActiveVgpus()* to query the currently active vGPU instances on the user-provided GPU Instance.
- Added *nvmlGpuInstanceGetVgpuSchedulerState()* to query the vGPU software scheduler state for the given GPU Instance.
- Added *nvmlGpuInstanceGetVgpuSchedulerLog()* to query the vGPU software scheduler logs for the given GPU Instance.
- Added *nvmlGpuInstanceGetVgpuTypeCreatablePlacements()* to query the creatable vGPU placement IDs of the vGPU type within a GPU Instance.
- Added *nvmlGpuInstanceSetVgpuHeterogeneousMode()* to enable or disable vGPU heterogeneous mode for the GPU Instance.
- Added *nvmlGpuInstanceGetVgpuHeterogeneousMode()* to query the vGPU heterogeneous mode for the GPU Instance.
- Updated *nvmlDeviceGetVgpuCapabilities()* to report whether GPU supports timesliced vGPU on MIG and whether MIG timesliced mode is enabled or not.
- Updated *nvmlDeviceSetVgpuCapabilities()* to set the MIG timesliced mode vGPU capability of a device.
- Updated *nvmlDeviceSetVgpuHeterogeneousMode()* to return *NVML_ERROR_NOT_SUPPORTED* when in MIG mode.

- Updated *nvmlDeviceGetVgpuHeterogeneousMode()* to return *NVML_ERROR_NOT_SUPPORTED* when in MIG mode.
- Updated *nvmlDeviceGetVgpuTypeCreatablePlacements()* to return *NVML_ERROR_NOT_SUPPORTED* when in MIG mode.
- Updated *nvmlDeviceGetVgpuSchedulerLog()* to return *NVML_ERROR_NOT_SUPPORTED* when in MIG mode.
- Updated *nvmlDeviceSetVgpuSchedulerState()* to return *NVML_ERROR_NOT_SUPPORTED* when in MIG mode.
- Updated *nvmlDeviceGetVgpuSchedulerState()* to return *NVML_ERROR_NOT_SUPPORTED* when in MIG mode.
- Added 3 new `NVML_FI_DEV_C2C_LINK_ERROR` field IDs:
  - *NVML_FI_DEV_C2C_LINK_ERROR_INTR*
  - *NVML_FI_DEV_C2C_LINK_ERROR_REPLAY*
  - *NVML_FI_DEV_C2C_LINK_ERROR_REPLAY_B2B*
- Added *NVML_FI_DEV_C2C_LINK_POWER_STATE* field ID.
- Added new CTXSW GPM Metrics.
- Added *nvmlDeviceGetHandleByUUIDV()* that supports both ASCII and binary format UUID to retrieve the device handle.
- Added 2 new `NVML_FI_DEV_POWER_SYNC_BALANCING` field IDs:
  - *NVML_FI_DEV_POWER_SYNC_BALANCING_FREQ*
  - *NVML_FI_DEV_POWER_SYNC_BALANCING_AF*
- Added 5 new Clock Event Reason Counters field IDs:
  - *NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_POWER_CAP*
  - *NVML_FI_DEV_CLOCKS_EVENT_REASON_SYNC_BOOST*
  - *NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_THERM_SLOWDOWN*
  - *NVML_FI_DEV_CLOCKS_EVENT_REASON_HW_THERM_SLOWDOWN*
  - *NVML_FI_DEV_CLOCKS_EVENT_REASON_HW_POWER_BRAKE_SLOWDOWN*
- Updated *nvmlDeviceGetMemoryErrorCounter()* to better account for transient vs. permanent errors.
- Added MIG profiles that can allocate all or none of Decoder, Encoder, JPEG, and OFA engines.

3.8. Changes between v565 and v570

The following new functionality is exposed on NVIDIA display drivers version 570 Production or later.

- Added field values for data related to Power Smoothing
- Added *nvmlDevicePowerSmoothingActivatePresetProfile()* to activate a specific Preset Profile for Power Smoothing
- Added *nvmlDevicePowerSmoothingSetState()* to enable/disable the Power Smoothing feature

- Added *nvmlDevicePowerSmoothingUpdatePresetProfileParam()* to update parameters to preset profiles for Power Smoothing
- Added new enums for fieldId NVML_FI_DEV_NVLINK_GET_STATE to expose INACTIVE, ACTIVE, and SLEEP state for a link
- Added *nvmlDeviceGetMarginTemperature()* to retrieve the thermal margin temperature (distance to nearest slowdown threshold).
- Added *nvmlDeviceGetNvlinkSupportedBwModes()* to get all supported Nvlink Bandwidth modes
- Added *nvmlDeviceGetNvlinkBwMode()* to get the current Nvlink Bandwidth mode
- Added *nvmlDeviceSetNvlinkBwMode()* to set the Nvlink Bandwidth mode
- Added MIG profiles with support for graphics.
- Added support for new recovery action - NVML_GPU_RECOVERY_ACTION_DRAIN_AND_RESET
- Deprecated nvml fieldIds NVML_FI_DEV_RESET_STATUS and NVML_FI_DEV_DRAIN_AND_RESET_STATUS. Usee NVML_FI_DEV_GET_GPU_RECOVERY_ACTION instead
- Added *nvmlDeviceGetDramEncryptionMode()* and *nvmlDeviceSetDramEncryptionMode()* to query and configure DRAM Encryption Mode
- Added 3 new flags to GPU Fabric Health Mask:
  - NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ROUTE_RECOVERY
  - NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ROUTE_UNHEALTHY
  - NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ACCESS_TIMEOUT_RECOVERY
- Added new counters for Nvlink5
- *NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_ERRORS* to get sum of the number of errors in each Nvlink packet
- *NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_BER* to get Effective BER for effective errors
- *NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_0* to 15 to get count of symbol errors that are corrected
- Swapped the values of field IDs *NVML_FI_DEV_IS_MIG_MODE_INDEPENDENT_MIG_QUERY_CAPABLE* and *NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_MAX* to fix backwards compatibility with v550.
- New revision of `nvmlPlatformInfo_t -- nvmlPlatformInfo_v2` has been added. In this version the following fields from v1 have been renamed
  - rackGuid to chassisSerialNumber
  - chassisPhysicalSlotNumber to slotNumber
  - computeSlotIndex to trayIndex
  - nodeIndex to hostId
- *nvmlPlatformInfo_v1* is deprecated and will be removed in subsequent releases
- Revert the fix for the issue where PCIe throughput (reported via *nvmlDeviceGetPcieThroughput()* and nvidia-smi -q) is 1000 times bigger than its actual value

3.9. Changes between v560 and v565

The following new functionality is exposed on NVIDIA display drivers version 565 Production or later.

- Fixed the ECC error count mismatch between nvidia-smi query output and NVML APIs, *nvmlDeviceGetMemoryErrorCounter()* and *nvmlDeviceGetFieldValues()*.
- Added new value NVML_CC_SYSTEM_CPU_CAPS_AMD_SNP_VTOM for CC CPU capability reporting.
- Added *nvmlDeviceGetCoolerInfo()* to retrieve a cooler's control signal characteristics and target that cooler cools.
- Added new value *NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV_SNP* for CC CPU capability reporting.
- Added *nvmlDeviceGetFanSpeedRPM()* to report the intended operating speed in rotations per minute (RPM) of the device's specified fan.
- Added *nvmlDeviceGetPerformanceModes()* to retrieve a performance modes string with all the performance modes defined for this device along with their associated GPU Clock and Memory Clock values.
- Added *nvmlDeviceGetCurrentClockFreqs()* to retrieve a string with the associated GPU Clock and Memory Clock values for the current pstate.
- Added *nvmlNvlinkVersion_t* enum to define NvLink Version.
- Added *nvmlDeviceGetPlatformInfo()* to retrieve the platform information of a device.
- Added new event type nvmlEventTypeGpuUnavailableError.
- Removed support for `nvmlDeviceGetNvLinkCrcLaneErrorCounter`, `nvmlDeviceGetNvLinkEccLaneErrorCounter`, and *nvmlDeviceGetNvLinkErrorCounter()* on Blackwell.
- Removed support for fieldIds *NVML_FI_DEV_NVLINK_ERROR_DL_REPLAY*, *NVML_FI_DEV_NVLINK_ERROR_DL_RECOVERY*, and *NVML_FI_DEV_NVLINK_ERROR_DL_CRC* on Blackwell.
- Added *nvmlVgpuInstanceGetRuntimeStateSize()* to get the vGPU runtime state size.
- Updated *nvmlDeviceGetVgpuTypeSupportedPlacements()* function to report both Heterogeneous and Homogeneous vGPU placements.
- Updated nvmlDeviceGetVgpuCapabilities to report the Homogeneous vGPU capability.
- Added new event type *nvmlEventTypeGpuRecoveryAction*.
- Added new fieldId to query GPU recovery action *NVML_FI_DEV_GET_GPU_RECOVERY_ACTION*.
- Deprecated fieldIds:
  - *NVML_FI_DEV_NVLINK_COUNT_VL15_DROPPED* to get Number of VL15 MADs dropped on a link in NVLink5
  - *NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE0* to get BER per lane for lane 0
  - *NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE1* to get BER per lane for lane 1
  - *NVML_FI_DEV_NVLINK_COUNT_RAW_BER* to get BER per link. Sum of all the raw errors per lane/Bits received per link
  - *NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_ERRORS* to get Sum of the number of errors in each Nvlink packet
  - *NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_BER* to get Effective BER for effective errors

3.10. Changes between v555 and v560

The following new functionality is exposed on NVIDIA display drivers version 560 Production or later.

- Added field values *NVML_FI_DEV_PCIE_OUTBOUND_ATOMICS_MASK* and *NVML_FI_DEV_PCIE_INBOUND_ATOMICS_MASK* for *nvmlDeviceGetFieldValues()*.
- Added field IDs *NVML_FI_DEV_RESET_STATUS* and *NVML_FI_DEV_DRAIN_AND_RESET_STATUS* which correspond to the nvidia-smi output.
- Added `NVML_DEVICE_ARCH_T23X` architecture type.
- Added *nvmlVgpuTypeGetBAR1Info()* to query the BAR1 information of a vGPU type.
- Added new event types, *nvmlEventTypeSingleBitEccErrorStorm*, *nvmlEventTypeDramRetirementEvent*, *nvmlEventTypeDramRetirementFailure*, *nvmlEventTypeNonFatalPoisonError* and *nvmlEventTypeFatalPoisonError*.
- Added *nvmlSystemGetDriverBranch()* to query the driver branch information.

3.11. Changes between v550 and v555

The following new functionality is exposed on NVIDIA display drivers version 555 Production or later.

- Added *nvmlDeviceGetClockOffsets()* to query min, max and current clock offset value on a Maxwell and later GPU for a specified clock. Note: *nvmlDeviceGetGpcClkVfOffset()*, *nvmlDeviceGetMemClkVfOffset()*, *nvmlDeviceGetGpcClkMinMaxVfOffset()* and *nvmlDeviceGetMemClkMinMaxVfOffset()* will be deprecated in a future release. Use *nvmlDeviceGetClockOffsets()* instead.
- Added *nvmlDeviceSetClockOffsets()* to control clock offset value on a Maxwell and later GPU for a specified clock. Note: *nvmlDeviceSetGpcClkVfOffset()* and *nvmlDeviceSetMemClkVfOffset()* will be deprecated in a future release. Use *nvmlDeviceSetClockOffsets()* instead.
- Added two new field IDs *NVML_FI_DEV_PCIE_COUNT_TX_BYTES* and *NVML_FI_DEV_PCIE_COUNT_RX_BYTES* for *nvmlDeviceGetFieldValues()*.
- Added new API `nvmlDeviceGetCapabilities` with the first capability bit `NVML_DEV_CAP_EGM` for Extended GPU Memory (EGM) capability.
- Added `multiGpuMode` display on CC enabled system via new API *nvmlSystemGetConfComputeSettings()* or "`nvidia-smi conf-compute --get-multigpu-mode`" or "`nvidia-smi conf-compute -mgm`".
- Added new field ID *NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_MAX* to get the Max Nvlink Power Threshold for a device.

3.12. Changes between v545 and v550

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

- Added *nvmlDeviceGetNumaNodeId()* to query the NUMA node of a GPU.
- Added new GPM metric ID *NVML_GPM_METRIC_NVOFA_1_UTIL* to *nvmlGpmMetricId_t*.
- Added new field ID *NVML_FI_DEV_IS_MIG_MODE_INDEPENDENT_MIG_QUERY_CAPABLE*, to check MIG query capable device irrespective of MIG mode.

- Deprecated *NVML_P2P_CAPS_INDEX_PROP* and added *NVML_P2P_CAPS_INDEX_PCI* to reflect the same P2P capability.
- Added *nvmlDeviceGetProcessesUtilizationInfo()* to retrieve the recent utilization and process ID for all running processes.
- Added new struct *nvmlProcessesUtilizationInfo_v1_t*, which includes the new utilization of NVJPG and NVOFA.
- Added *nvmlDeviceGetVgpuInstancesUtilizationInfo()* to retrieve the recent utilization for vGPU instances running on a physical GPU.
- Added *nvmlDeviceGetVgpuProcessesUtilizationInfo()* to retrieve the recent utilization for processes running on vGPU instances on a physical GPU.
- Added *nvmlDeviceSetVgpuHeterogeneousMode()* to enable or disable vGPU heterogenous mode for the device.
- Added *nvmlDeviceGetVgpuHeterogeneousMode()* to query the vGPU heterogenous mode for the device.
- Added *nvmlVgpuInstanceGetPlacementId()* to query placement ID of the active vGPU instance.
- Added *nvmlDeviceGetVgpuTypeSupportedPlacements()* to query the supported vGPU placement IDs of a vGPU type.
- Added *nvmlDeviceGetVgpuTypeCreatablePlacements()* to query the creatable vGPU placement IDs of a vGPU type.
- Added support to display confidential compute protected memory along with `fb` and `bar1` in `nvidia-smi pmon` and `dmon` commands.
- Added *nvmlDeviceGetGpuFabricInfoV()* to query GPU Fabric Probe Info for the device.
- Deprecated *nvmlDeviceGetGpuFabricInfo()*. This function should not be used, and will be removed in a future release. Use *nvmlDeviceGetGpuFabricInfoV()* instead.
- Modified *nvmlDeviceGetGpuInstanceProfileInfo()* and *nvmlDeviceGetGpuInstancePossiblePlacements_v2()* to no longer require MIG being enabled.
- Added new encoder type *NVML_ENCODER_QUERY_AV1* and *NVML_ENCODER_QUERY_UNKNOWN* to enumeration *nvmlEncoderType_t*.
- Added *nvmlSystemSetConfComputeKeyRotationThresholdInfo()* to set confidential compute key rotation threshold.
- Added *nvmlSystemGetConfComputeKeyRotationThresholdInfo()* to query confidential compute key rotation threshold detail.
- Added *nvmlDeviceSetVgpuCapabilities()* to set the desirable vGPU capability of a device.

3.13. Changes between v535 and v545

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

- Added a new error code *NVML_ERROR_GPU_NOT_FOUND* to be returned if no supported GPUS are found during initialization.
- In *nvmlGpuFabricInfo_v2_t*, `partitionId` has been renamed to `cliqueId`.

- Added new versioned structs *nvmlGpuInstanceProfileInfo_v3_t* and *nvmlComputeInstanceProfileInfo_v3_t*.
- Added *nvmlDeviceGetLastBBXFlushTime()* for retrieving the timestamp and duration of the latest flush of the BBX object to the inforom storage.
- Added *NVML_POWER_SCOPE_MEMORY* to report out power usage for GPU Memory.
- Added *nvmlDeviceGetPciInfo_v3()* which expands *nvmlDeviceGetPciInfo* to also report PCI base and sub classcodes.
- Added new struct *nvmlPciInfoExt_v1_t*, which is used in *nvmlDeviceGetPciInfoExt()*.
- Added *nvmlDeviceGetRunningProcessDetailList()* API to get information about Compute, Graphics or MPS-Compute processes running on a GPU with protected memory usage info.

3.14. Changes between v530 and v535

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

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

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

3.15. Changes between v525 and v530

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

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

3.16. Changes between v520 and v525

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

- Added `nvmlDeviceGetPcieAtomicCaps` to report PCIe atomic capabilities.

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

3.17. Changes between v515 and v520

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

- Added *`nvmlDeviceGetMemClkVfOffset()`* API to report the MemClk VF offset value.
- Added *`nvmlDeviceSetMemClkVfOffset()`* API to set the MemClk VF offset value.
- Added *`nvmlDeviceGetMemClkMinMaxVfOffset()`* API to report the Memory clock min and max VF offset that user can set for a specified GPU.
- Added *`nvmlDeviceGetTargetFanSpeed()`* API to report the intended target speed of the device's specified fan.
- Added *`nvmlDeviceGetGpcClkMinMaxVfOffset()`* API to report the Graphics clock min and max VF offset that user can set for a specified GPU.
- Added *`nvmlGpmMetricsGet()`* to calculate GPM metrics from two GPM samples.
- Added *`nvmlGpmSampleFree()`* to free allocated GPM sample.
- Added *`nvmlGpmSampleAlloc()`* to allocate a GPM sample.
- Added *`nvmlGpmSampleGet()`* to retrieve a GPM snapshot.
- Added *`nvmlGpmQueryDeviceSupport()`* to query whether a device supports GPM
- Added `nvmlDeviceGetSupportedPowerModes` API to report the GPU's supported power mode mask.
- Added `nvmlDeviceGetPowerMode` API to report the GPU's current power mode.
- Added `nvmlDeviceSetPowerMode` API to set the new power mode.
- Added *`nvmlDeviceGetFanControlPolicy_v2()`* API to report the control policy for a specified GPU fan.
- Added *`nvmlDeviceSetFanControlPolicy()`* API to set the control policy for a specified GPU fan.

3.18. Changes between v510 and v515

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

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

3.19. Changes between v495 and v510

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

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

3.20. Changes between v465 and v470

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

- Added new MIG GPU instance profile NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV1.

- Added *nvmlDeviceGetGpuInstancePossiblePlacements_v2()*. The previous version of the API will not support the profiles with possible placements greater than its total capacity, such as NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV1.

3.21. Changes between v460 and v465

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

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

3.22. Changes between v450 and v460

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

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

3.23. Changes between v445 and v450

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

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

3.24. Changes between v418 and v445

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

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

3.25. Changes between v361 and v418

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

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

3.26. Changes between v349 and v361

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

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

3.27. Changes between v346 and v349

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

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

3.28. Changes between v340 and v346

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

- Added `nvmlDeviceGetGraphicsRunningProcesses_v2` to get information about Graphics Processes running on a device.
- Added *nvmlDeviceGetPcieReplayCounter()* to get PCI replay counters.
- Added *nvmlDeviceGetPcieThroughput()* to get PCI utilization information.
- Discontinued Perl bindings support.

3.29. Changes between NVML v331 and v340

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

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

3.30. Changes between NVML v5.319 Update and v331

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

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

3.31. 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()*.

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

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

- Added _v2 versions of *nvmlDeviceGetHandleByIndex* and *nvmlDeviceGetCount* that also count devices not accessible by current user
  - *nvmlDeviceGetHandleByIndex_v2()* (default) can also return NVML_ERROR_NO_PERMISSION
- Added *nvmlInit_v2()* and *nvmlDeviceGetHandleByIndex_v2()* that is safer and thus recommended function for initializing the library
  - *nvmlInit_v2()* lazily initializes only requested devices (queried with nvmlDeviceGetHandle*)
  - nvml.h defines *nvmlInit_v2()* and *nvmlDeviceGetHandleByIndex_v2()* as default functions
- Added *nvmlDeviceGetIndex()*
- Added NVML_ERROR_GPU_IS_LOST to report GPUs that have fallen off the bus.
  - All NVML device APIs can return this error code, as a GPU can fall off the bus at any time.
- Added new class of APIs for gathering process statistics (`nvmlAccountingStats`)
- Application Clocks are no longer supported on GPU's from Quadro product line
- Added APIs to support dynamic page retirement. See *nvmlDeviceGetRetiredPages()* and *nvmlDeviceGetRetiredPagesPendingStatus()*
- Renamed *nvmlClocksThrottleReasonUserDefinedClocks* to *nvmlClocksThrottleReasonApplicationsClocksSetting*. Old name is deprecated and can be removed in one of the next major releases.
- Added *nvmlDeviceGetDisplayActive()* and updated documentation to clarify how it differs from *nvmlDeviceGetDisplayMode()*

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

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

- Added *nvmlDeviceGetGpuOperationMode()* and *nvmlDeviceSetGpuOperationMode()*.

3.34. Changes between NVML v3.295 and v4.304 RC

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

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

3.35. Changes between NVML v2.285 and v3.295

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

- Deprecated *nvmlDeviceGetHandleBySerial()* in favor of newly added *nvmlDeviceGetHandleByUUID()*.
- Marked the input parameters of *nvmlDeviceGetHandleBySerial()*, *nvmlDeviceGetHandleByUUID()* and *nvmlDeviceGetHandleByPciBusId* as const.
- Added *nvmlDeviceOnSameBoard()*.
- Added `nvmlConstants` defines.

- Added *nvmlDeviceGetMaxPcieLinkGeneration()*, *nvmlDeviceGetMaxPcieLinkWidth()*, *nvmlDeviceGetCurrPcieLinkGeneration()*,:cpp:func:*nvmlDeviceGetCurrPcieLinkWidth*.
- Format change of *nvmlDeviceGetUUID()* output to match the UUID standard. This function will return a different value.
- *nvmlDeviceGetDetailedEccErrors()* will report zero for unsupported ECC error counters when a subset of ECC error counters are supported.

3.36. 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`.
  - Fixed error message when querying pending driver model with `nvmlDeviceGetDriverModel` on non-Admin user account.
  - Fixed problem when *nvmlDeviceGetDisplayMode()* wouldn't update.
- Added API *nvmlDeviceGetVbiosVersion()* function to report VBIOS version.
- Added pciSubSystemId to *nvmlPciInfo_t* struct.
- Added API *nvmlErrorString()* function to convert error code to string.
- Updated docs to indicate we support M2075 and C2075.
- Added API *nvmlSystemGetHicVersion()* function to report HIC firmware version.
- Added NVML versioning support
  - Functions that changed API and/or size of structs have appended versioning suffix (e.g., nvmlDeviceGetPciInfo_v2). Appropriate C defines have been added that map old function names to the newer version of the function.
- Added support for concurrent library usage by multiple libraries.
- Added API *nvmlDeviceGetMaxClockInfo()* function for reporting device's clock limits.
- Added new error code NVML_ERROR_DRIVER_NOT_LOADED used by *nvmlInit*.
- Extended *nvmlPciInfo_t* struct with new field: sub system id.
- Added NVML support on Windows guest account.
- Changed format of pciBusId string (to XXXX:XX:XX.X) of *nvmlPciInfo_t*.
- Parsing of busId in *nvmlDeviceGetHandleByPciBusId* is less restrictive. You can pass 0:2:0.0 or 0000:02:00 and other variations.
- Added API for events waiting for GPU events (Linux only) see docs of `nvmlEvents`.
- Added API `nvmlDeviceGetComputeRunningProcesses_v2` and *nvmlSystemGetProcessName()* functions for looking up currently running compute applications.
- Deprecated *nvmlDeviceGetPowerState()* in favor of *nvmlDeviceGetPerformanceState()*.

Chapter 4. Deprecation and/or Removal Notices for the NVML Library

This chapter lists the NVML functions and data structures marked for deprecation and/or removal. Starting with CUDA 13.1, deprecated functions will generate compiler warnings. Removed functions will return the NVML error code *NVML_ERROR_DEPRECATED*.

4.1. CUDA 13.0

The following functions are deprecated starting with CUDA 13.0; they will be removed in CUDA 14.0:

- *nvmlDeviceSetApplicationsClocks()*
- *nvmlDeviceGetApplicationsClock()*
- *nvmlDeviceGetDefaultApplicationsClock()*
- *nvmlDeviceResetApplicationsClocks()*
- *nvmlDeviceGetViolationStatus()*
- *nvmlVgpuInstanceGetLicenseStatus()*
- *nvmlDeviceResetNvLinkUtilizationCounter()*
- *nvmlDeviceFreezeNvLinkUtilizationCounter()*
- *nvmlDeviceGetNvLinkUtilizationCounter()*
- *nvmlDeviceGetNvLinkUtilizationControl()*
- *nvmlDeviceSetNvLinkUtilizationControl()*
- *nvmlDeviceSetMemClkVfOffset()*
- *nvmlDeviceSetGpcClkVfOffset()*
- *nvmlDeviceGetGpuFabricInfo()*
- *nvmlDeviceGetDetailedEccErrors()*
- *nvmlDeviceGetPowerManagementMode()*
- *nvmlDeviceGetPowerState()*
- *nvmlDeviceGetSupportedClocksThrottleReasons()*
- *nvmlDeviceGetCurrentClocksThrottleReasons()*
- *nvmlDeviceGetTemperature()*

- *nvmlDeviceGetHandleBySerial()*

The following data structures are deprecated starting with CUDA 13.0:

- *nvmlGpuFabricInfo_v2_t*

Chapter 5. NVML API Reference

5.1. Namespace hierarchy

- struct *nvmlAccountingStats_t*
- struct *nvmlAccountingStats_v2_t*
- struct *nvmlActiveVgpuInstanceInfo_v1_t*
- struct *nvmlAdaptiveTgpModeInfo_v1_t*
- struct *nvmlBAR1Memory_t*
- struct *nvmlBBXTimeData_v1_t*
- struct *nvmlBridgeChipHierarchy_t*
- struct *nvmlBridgeChipInfo_t*
- struct *nvmlC2cModeInfo_v1_t*
- struct *nvmlCPERCursor_v1_t*
- struct *nvmlClkMonFaultInfo_t*
- struct *nvmlClkMonStatus_t*
- struct *nvmlClockOffset_v1_t*
- struct *nvmlComputeInstanceInfo_t*
- struct *nvmlComputeInstancePlacement_t*
- struct *nvmlComputeInstanceProfileInfo_t*
- struct *nvmlComputeInstanceProfileInfo_v2_t*
- struct *nvmlComputeInstanceProfileInfo_v3_t*
- struct *nvmlConfComputeGetKeyRotationThresholdInfo_v1_t*
- struct *nvmlConfComputeGpuAttestationReport_t*
- struct *nvmlConfComputeGpuCertificate_t*
- struct *nvmlConfComputeMemSizeInfo_t*
- struct *nvmlConfComputeSetKeyRotationThresholdInfo_v1_t*
- struct *nvmlConfComputeSystemCaps_t*
- struct *nvmlConfComputeSystemState_t*
- struct *nvmlCoolerInfo_v1_t*

- struct *nvmlCoreRailMetrics_t*
- struct *nvmlDeviceAddressingMode_v1_t*
- struct *nvmlDeviceAttributes_t*
- struct *nvmlDeviceCapabilities_v1_t*
- struct *nvmlDeviceCurrentClockFreqs_v1_t*
- struct *nvmlDevicePerfModes_v1_t*
- struct *nvmlDevicePowerMizerModes_v1_t*
- struct *nvmlDramEncryptionInfo_v1_t*
- struct *nvmlEccBankRemapperHistogram_v1_t*
- struct *nvmlEccBankRemapperStatus_v1_t*
- struct *nvmlEccErrorCounts_t*
- struct *nvmlEccSramErrorStatus_v1_t*
- struct *nvmlEccSramUniqueUncorrectedErrorCounts_v1_t*
- struct *nvmlEccSramUniqueUncorrectedErrorEntry_v1_t*
- struct *nvmlEncoderSessionInfo_t*
- struct *nvmlEventData_t*
- struct *nvmlEventSetGetContextCount_v1_t*
- struct *nvmlEventSetGetContextData_v1_t*
- struct *nvmlEventSetGetContextInfo_v1_t*
- struct *nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t*
- struct *nvmlEventSetWait_v3_t*
- struct *nvmlExcludedDeviceInfo_t*
- struct *nvmlFBCSessionInfo_t*
- struct *nvmlFBCStats_t*
- struct *nvmlFanSpeedInfo_v1_t*
- struct *nvmlFieldValue_t*
- struct *nvmlGetCPER_v1_t*
- struct *nvmlGetMemoryLimits_v1_t*
- struct *nvmlGpmMetric_t*
- struct *nvmlGpmMetricsGet_t*
- struct *nvmlGpmSupport_t*
- struct *nvmlGpuDynamicPstatesInfo_t*
- struct *nvmlGpuFabricClique_v1_t*
- struct *nvmlGpuFabricInfo_t*
- struct *nvmlGpuFabricInfo_v2_t*
- struct *nvmlGpuFabricInfo_v3_t*

- struct *nvmlGpuFabricInfo_v4_t*
- struct *nvmlGpuInstanceInfo_t*
- struct *nvmlGpuInstancePlacement_t*
- struct *nvmlGpuInstanceProfileInfo_t*
- struct *nvmlGpuInstanceProfileInfo_v2_t*
- struct *nvmlGpuInstanceProfileInfo_v3_t*
- struct *nvmlGpuOperationalEventConfig_v1_t*
- struct *nvmlGpuThermalSettings_t*
- struct *nvmlGpuUptime_v1_t*
- struct *nvmlGridLicensableFeature_t*
- struct *nvmlGridLicensableFeatures_t*
- struct *nvmlGridLicenseExpiry_t*
- struct *nvmlHostname_v1_t*
- struct *nvmlHwbcEntry_t*
- struct *nvmlLedState_t*
- struct *nvmlMarginTemperature_v1_t*
- struct *nvmlMask255_t*
- struct *nvmlMemory_t*
- struct *nvmlMemory_v2_t*
- struct *nvmlNvLinkInfo_v1_t*
- struct *nvmlNvLinkInfo_v2_t*
- struct *nvmlNvLinkInfo_v3_t*
- struct *nvmlNvLinkPowerThres_t*
- struct *nvmlNvLinkUtilizationControl_t*
- struct *nvmlNvlinkFirmwareInfo_t*
- struct *nvmlNvlinkFirmwareVersion_t*
- struct *nvmlNvlinkGetBwMode_v1_t*
- struct *nvmlNvlinkGetRemapTableInfo_v1_t*
- struct *nvmlNvlinkLockRemapTable_v1_t*
- struct *nvmlNvlinkNvlePrivateInfo_v1_t*
- struct *nvmlNvlinkSetBwModeAsync_v1_t*
- struct *nvmlNvlinkSetBwMode_v1_t*
- struct *nvmlNvlinkSetNvleReady_v1_t*
- struct *nvmlNvlinkSetupNvleEncryptionKey_v1_t*
- struct *nvmlNvlinkSupportedBwModes_v1_t*
- struct *nvmlNvlinkTelemetrySample_v1_t*

- struct *nvmlNvlinkTelemetrySamples_v1_t*
- struct *nvmlObservedMetrics_t*
- struct *nvmlPRMCounterInput_v1_t*
- struct *nvmlPRMCounterList_v1_t*
- struct *nvmlPRMCounterValue_v1_t*
- struct *nvmlPRMCounter_v1_t*
- struct *nvmlPRMTLV_v1_t*
- struct *nvmlPSUInfo_t*
- struct *nvmlPciInfoExt_v1_t*
- struct *nvmlPciInfo_t*
- struct *nvmlPdi_v1_t*
- struct *nvmlPerfMetricControllerSample_t*
- struct *nvmlPerfMetricsDlppc2xSample_t*
- struct *nvmlPerfMetricsPfpp1xSample_t*
- struct *nvmlPerfMetricsSample_t*
- struct *nvmlPerfMetricsSamples_v1_t*
- struct *nvmlPlatformInfo_v1_t*
- struct *nvmlPlatformInfo_v2_t*
- struct *nvmlPmgrPwrTuple_t*
- struct *nvmlPowerSmoothingProfile_v1_t*
- struct *nvmlPowerSmoothingState_v1_t*
- struct *nvmlPowerValue_v2_t*
- struct *nvmlProcessDetailList_v1_t*
- struct *nvmlProcessDetail_v1_t*
- struct *nvmlProcessInfo_t*
- struct *nvmlProcessInfo_v1_t*
- struct *nvmlProcessUtilizationInfo_v1_t*
- struct *nvmlProcessUtilizationSample_t*
- struct *nvmlProcessesUtilizationInfo_v1_t*
- struct *nvmlPwrModelMetricsDlppm1xDramclkEstimates_t*
- struct *nvmlPwrModelMetricsDlppm1xPerf_t*
- struct *nvmlPwrModelMetricsDlppm1x_t*
- struct *nvmlPwrModelMetricsPfpp1x_t*
- struct *nvmlPwrModelMetricsSamplePfpp1x_t*
- struct *nvmlPwrModelOperatingPointPfpp1x_t*
- struct *nvmlRailMetrics_t*

- struct *nvmlRemappedRowsInfo_v2_t*
- struct *nvmlRepairStatus_v1_t*
- struct *nvmlRowRemapperHistogramValues_t*
- struct *nvmlRusdSettings_v1_t*
- struct *nvmlSample_t*
- struct *nvmlSetMemoryLimits_v1_t*
- struct *nvmlSystemConfComputeSettings_v1_t*
- struct *nvmlSystemConfComputeSettings_v2_t*
- struct *nvmlSystemDriverBranchInfo_v1_t*
- struct *nvmlSystemEventData_v1_t*
- struct *nvmlSystemEventSetCreateRequest_v1_t*
- struct *nvmlSystemEventSetFreeRequest_v1_t*
- struct *nvmlSystemEventSetWaitRequest_v1_t*
- struct *nvmlSystemRegisterEventRequest_v1_t*
- struct *nvmlTemperature_v1_t*
- struct *nvmlUUID_v1_t*
- struct *nvmlUnitFanInfo_t*
- struct *nvmlUnitFanSpeeds_t*
- struct *nvmlUnitInfo_t*
- struct *nvmlUnrepairableMemoryStatus_v1_t*
- struct *nvmlUtilization_t*
- struct *nvmlVgpuCreatablePlacementInfo_v1_t*
- struct *nvmlVgpuHeterogeneousMode_v1_t*
- struct *nvmlVgpuInstanceUtilizationInfo_v1_t*
- struct *nvmlVgpuInstanceUtilizationSample_t*
- struct *nvmlVgpuInstancesUtilizationInfo_v1_t*
- struct *nvmlVgpuLicenseExpiry_t*
- struct *nvmlVgpuLicenseInfo_t*
- struct *nvmlVgpuMetadata_t*
- struct *nvmlVgpuPgpuCompatibility_t*
- struct *nvmlVgpuPgpuMetadata_t*
- struct *nvmlVgpuPlacementId_v1_t*
- struct *nvmlVgpuPlacementList_v1_t*
- struct *nvmlVgpuPlacementList_v2_t*
- struct *nvmlVgpuProcessUtilizationInfo_v1_t*
- struct *nvmlVgpuProcessUtilizationSample_t*

- struct *nvmlVgpuProcessesUtilizationInfo_v1_t*
- struct *nvmlVgpuRuntimeState_v1_t*
- struct *nvmlVgpuSchedulerCapabilities_t*
- struct *nvmlVgpuSchedulerGetState_t*
- struct *nvmlVgpuSchedulerLogEntry_t*
- struct *nvmlVgpuSchedulerLogEntry_v2_t*
- struct *nvmlVgpuSchedulerLogInfo_v1_t*
- struct *nvmlVgpuSchedulerLogInfo_v2_t*
- struct *nvmlVgpuSchedulerLog_t*
- struct *nvmlVgpuSchedulerSetState_t*
- struct *nvmlVgpuSchedulerStateInfo_v1_t*
- struct *nvmlVgpuSchedulerStateInfo_v2_t*
- struct *nvmlVgpuSchedulerState_v1_t*
- struct *nvmlVgpuSchedulerState_v2_t*
- struct *nvmlVgpuTypeBar1Info_v1_t*
- struct *nvmlVgpuTypeIdInfo_v1_t*
- struct *nvmlVgpuTypeMaxInstance_v1_t*
- struct *nvmlVgpuVersion_t*
- struct *nvmlViolationTime_t*
- struct *nvmlWorkloadPowerProfileCurrentProfiles_v1_t*
- struct *nvmlWorkloadPowerProfileInfo_v1_t*
- struct *nvmlWorkloadPowerProfileProfilesInfo_v1_t*
- struct *nvmlWorkloadPowerProfileRequestedProfiles_v1_t*
- struct *nvmlWorkloadPowerProfileUpdateProfiles_v1_t*

5.2. API contents

5.2.1. Groups

5.2.1.1 Accounting Statistics

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.

Note

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

5.2.1.1.1 Functions

nvmlReturn_t *nvmlDeviceClearAccountingPids*(nvmlDevice_t device)
Clears accounting information about all processes that have already terminated.

nvmlReturn_t *nvmlDeviceGetAccountingBufferSize*(nvmlDevice_t device, unsigned int *bufferSize)
Returns the number of processes that the circular buffer with accounting pids can hold.

nvmlReturn_t *nvmlDeviceGetAccountingMode*(nvmlDevice_t device, nvmlEnableState_t *mode)
Queries the state of per process accounting mode.

nvmlReturn_t *nvmlDeviceGetAccountingPids*(nvmlDevice_t device, unsigned int *count, unsigned int *pids)
Queries list of processes that can be queried for accounting stats.

nvmlReturn_t *nvmlDeviceGetAccountingStats*(nvmlDevice_t device, unsigned int pid, nvmlAccountingStats_t *stats)
Queries process's accounting stats.

nvmlReturn_t *nvmlDeviceGetAccountingStats_v2*(nvmlDevice_t device, nvmlAccountingStats_v2_t *stats)
Queries process's accounting stats (v2).

nvmlReturn_t *nvmlDeviceSetAccountingMode*(nvmlDevice_t device, nvmlEnableState_t mode)
Enables or disables per process accounting.

5.2.1.1.2 Structs

nvmlAccountingStats_t
Describes accounting statistics of a process.

nvmlAccountingStats_v2_t
Describes accounting statistics (v2) of a process.

5.2.1.1.3 Functions

nvmlReturn_t **nvmlDeviceClearAccountingPids**(*nvmlDevice_t* device)

Clears accounting information about all processes that have already terminated.

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

See *nvmlDeviceGetAccountingMode* See *nvmlDeviceGetAccountingStats* See *nvmlDeviceSetAccountingMode*

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

nvmlReturn_t **nvmlDeviceGetAccountingBufferSize**(
    nvmlDevice_t device, unsigned int *bufferSize,
)

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

For Kepler or newer fully supported devices.

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

See also

nvmlDeviceGetAccountingStats

See also

nvmlDeviceGetAccountingPids

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

nvmlReturn_t **nvmlDeviceGetAccountingMode**(
    nvmlDevice_t device, *nvmlEnableState_t* *mode,
)

Queries the state of per process accounting mode.

For Kepler or newer fully supported devices.

See *nvmlDeviceGetAccountingStats* for more details. See *nvmlDeviceSetAccountingMode*

Note

In the MPS locality-domain view, this API returns the physical parent's accounting mode as reported by MPS. See *MPS Locality Domain Device Enumeration*.

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

nvmlReturn_t **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 or newer fully supported devices.

To query the number of processes under Accounting Mode, call this function with *count = 0 and pids=NULL. The return code will be NVML_ERROR_INSUFFICIENT_SIZE with an updated count value indicating the number of processes.

For more details see *nvmlDeviceGetAccountingStats*.

See also

nvmlDeviceGetAccountingBufferSize

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 *device* doesn't support this feature or accounting mode is disabled or on vGPU host.
- *NVML_ERROR_INSUFFICIENT_SIZE* if *count* is too small (*count* is set to expected value)
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

Queries process's accounting stats.

For Kepler or newer fully supported devices.

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

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

See also

nvmlDeviceGetAccountingBufferSize

Note

Accounting Mode needs to be on. See *nvmlDeviceGetAccountingMode*.

Note

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

Note

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 *device* doesn't support this feature or accounting mode is disabled or on vGPU host.
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetAccountingStats_v2**(
    nvmlDevice_t device, *nvmlAccountingStats_v2_t* *stats,
)

Queries process's accounting stats (v2).

For Kepler or newer fully supported devices.

Accounting stats (v2) capture GPU utilization and other statistics across the lifetime of a process. Accounting stats (v2) can be queried during life time of the process and after its termination. The time field in *nvmlAccountingStats_v2_t* is reported as 0 during the lifetime of the process and updated to actual running time after its termination. Accounting stats (v2) are kept in a circular buffer, newly created processes overwrite information about old processes.

See *nvmlAccountingStats_v2_t* for description of each returned metric. List of processes that can be queried can be retrieved from *nvmlDeviceGetAccountingPids*.

See also

nvmlDeviceGetAccountingBufferSize

Note

Accounting Mode needs to be on. See *nvmlDeviceGetAccountingMode*.

Note

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

Note

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
- **`stats`** – Reference in which to return the process's accounting stats (v2)

Returns

- *NVML_SUCCESS* if stats (v2) have been successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *stats* are NULL
- *NVML_ERROR_NOT_FOUND* if process stats were not found
- *NVML_ERROR_NOT_SUPPORTED* if *device* doesn't support this feature or accounting mode is disabled or on vGPU host.
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **`nvmlDeviceSetAccountingMode`**(
nvmlDevice_t device, *nvmlEnableState_t* mode,
)

Enables or disables per process accounting.

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

See *nvmlDeviceGetAccountingMode* See *nvmlDeviceGetAccountingStats* See *nvmlDeviceClearAccountingPids*

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.

Note

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

Note

Disabling accounting clears all accounting pids information.

Note

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

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

5.2.1.2 CPER (Common Platform Error Record)

Types and API for retrieving CPER data.

5.2.1.2.1 Macros

NVML_CPER_CURSOR_HANDLE_INIT
Initialize *nvmlCPERCursorHandle_t* to this value before first use in any CPER API.

5.2.1.2.2 Enumerations

nvmlCPERType_t
Bitmask of CPER record types.

5.2.1.2.3 Structs

nvmlCPERCursor_v1_t
CPER query and cursor parameters.

nvmlGetCPER_v1_t
Input/output structure for *nvmlSystemGetCPER_v1* .

5.2.1.2.4 Typedefs

nvmlCPERCursorHandle_t
Opaque handle to a CPER read position.

5.2.1.2.5 Macros

NVML_CPER_CURSOR_HANDLE_INIT
Initialize *nvmlCPERCursorHandle_t* to this value before first use in any CPER API.

5.2.1.2.6 Enumerations

enum **nvmlCPERType_t**
Bitmask of CPER record types.
Multiple values may be combined to request records from several sources in one call.
Values:

enumerator **NVML_CPER_ACCESS_TYPE_GPU**
Access GPU CPER records.

5.2.1.2.7 Typedefs

typedef unsigned long long **nvmlCPERCursorHandle_t**
Opaque handle to a CPER read position.

5.2.1.3 Confidential Computing definitions

5.2.1.3.1 Macros

NVML_CC_ACCEPTING_CLIENT_REQUESTS_FALSE
Confidential Compute GPUs/System Ready State values.

NVML_CC_ACCEPTING_CLIENT_REQUESTS_TRUE
Client requests are accepted.

NVML_CC_CEC_ATTESTATION_REPORT_NOT_PRESENT
CEC attestation report is not present.

NVML_CC_CEC_ATTESTATION_REPORT_PRESENT
CEC attestation report is present.

NVML_CC_GPU_ATTESTATION_REPORT_SIZE
Size of the attestation report.

NVML_CC_GPU_CEC_ATTESTATION_REPORT_SIZE
Size of the CEC attestation report.

NVML_CC_GPU_CEC_NONCE_SIZE
GPU Attestation Report.

NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MAX
Maximum attacker advantage for key rotation threshold.

NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MIN
Minimum attacker advantage for key rotation threshold.

NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV
AMD SEV CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV_SNP
AMD SEV-SNP CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV_TIO
AMD SEV-TIO CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_AMD_SNP_VTOM
AMD SNP-VTOM CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_ARM_CCA
ARM CCA CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_INTEL_TDX
Intel TDX CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_INTEL_TDX_CONNECT
Intel TDX Connect CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_NONE
Confidential Compute CPU Capabilities values.

NVML_CC_SYSTEM_DEVTOOLS_MODE_OFF
Confidential Compute DevTools Mode values.

NVML_CC_SYSTEM_DEVTOOLS_MODE_ON
DevTools mode is on.

NVML_CC_SYSTEM_ENVIRONMENT_PROD
Environment is production.

NVML_CC_SYSTEM_ENVIRONMENT_SIM
Environment is simulation.

NVML_CC_SYSTEM_ENVIRONMENT_UNAVAILABLE
Confidential Compute Environment values.

NVML_CC_SYSTEM_FEATURE_BMSAI_ENABLED
Feature is enabled for Baremetal Secure AI.

NVML_CC_SYSTEM_FEATURE_DISABLED
Confidential Compute Feature Status values.

NVML_CC_SYSTEM_FEATURE_ENABLED
Feature is enabled.

NVML_CC_SYSTEM_GPUS_CC_BMSAI_CAPABLE
GPUs are Baremetal Secure AI capable.

NVML_CC_SYSTEM_GPUS_CC_CAPABLE
GPUs are confidential compute capable.

NVML_CC_SYSTEM_GPUS_CC_NOT_CAPABLE
Confidenial Compute GPU Capabilities values.

NVML_CC_SYSTEM_MULTIGPU_NONE
Confidential Compute Multigpu mode values.

NVML_CC_SYSTEM_MULTIGPU_NVLE
Multi-GPU mode is NVLE.

NVML_CC_SYSTEM_MULTIGPU_PROTECTED_PCIE
Multi-GPU mode is protected PCIe.

NVML_CC_SYSTEM_TDISP_STATE_DISABLED
Confidential Compute TDISP state values.

NVML_CC_SYSTEM_TDISP_STATE_NOT_READY
The TDIs are UNLOCKED, i.e. TDISP FW-only mode.

NVML_CC_SYSTEM_TDISP_STATE_READY
The TDIs completed the TSM handshake and reached RUN, i.e. TDISP secure mode.

NVML_GPU_ATTESTATION_CERT_CHAIN_SIZE
Size of the attestation certificate chain.

NVML_GPU_CERT_CHAIN_SIZE
GPU Certificate Details.

nvmlConfComputeGetKeyRotationThresholdInfo_v1
Version macro for *nvmlConfComputeGetKeyRotationThresholdInfo_v1_t* .

nvmlConfComputeSetKeyRotationThresholdInfo_v1
Version macro for *nvmlConfComputeSetKeyRotationThresholdInfo_v1_t* .

nvmlSystemConfComputeSettings_v1
Version macro for *nvmlSystemConfComputeSettings_v1_t* .

nvmlSystemConfComputeSettings_v2
Version macro for *nvmlSystemConfComputeSettings_v2_t* .

5.2.1.3.2 Structs

nvmlConfComputeGetKeyRotationThresholdInfo_v1_t

nvmlConfComputeGpuAttestationReport_t

nvmlConfComputeGpuCertificate_t

nvmlConfComputeMemSizeInfo_t
Protected memory size.

nvmlConfComputeSetKeyRotationThresholdInfo_v1_t

nvmlConfComputeSystemCaps_t

nvmlConfComputeSystemState_t

nvmlSystemConfComputeSettings_v1_t
Confidential Compute System settings.

nvmlSystemConfComputeSettings_v2_t
Confidential Compute System settings, version 2.

5.2.1.3.3 Typedefs

nvmlConfComputeGetKeyRotationThresholdInfo_t

nvmlConfComputeSetKeyRotationThresholdInfo_t

nvmlSystemConfComputeSettings_t

5.2.1.3.4 Macros

NVML_CC_ACCEPTING_CLIENT_REQUESTS_FALSE
Confidential Compute GPUs/System Ready State values.

Client requests are not accepted.

NVML_CC_ACCEPTING_CLIENT_REQUESTS_TRUE
Client requests are accepted.

NVML_CC_CEC_ATTESTATION_REPORT_NOT_PRESENT
CEC attestation report is not present.

NVML_CC_CEC_ATTESTATION_REPORT_PRESENT
CEC attestation report is present.

NVML_CC_GPU_ATTESTATION_REPORT_SIZE
Size of the attestation report.

NVML_CC_GPU_CEC_ATTESTATION_REPORT_SIZE
Size of the CEC attestation report.

NVML_CC_GPU_CEC_NONCE_SIZE
GPU Attestation Report.

Size of the CEC nonce.

NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MAX
Maximum attacker advantage for key rotation threshold.

NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MIN
Minimum attacker advantage for key rotation threshold.

NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV
AMD SEV CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV_SNP
AMD SEV-SNP CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV_TIO
AMD SEV-TIO CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_AMD_SNP_VTOM
AMD SNP-VTOM CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_ARM_CCA
ARM CCA CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_INTEL_TDX
Intel TDX CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_INTEL_TDX_CONNECT
Intel TDX Connect CPU capability.

NVML_CC_SYSTEM_CPU_CAPS_NONE
Confidential Compute CPU Capabilities values.
No confidential compute CPU capability.

NVML_CC_SYSTEM_DEVTOOLS_MODE_OFF
Confidential Compute DevTools Mode values.
DevTools mode is off.

NVML_CC_SYSTEM_DEVTOOLS_MODE_ON
DevTools mode is on.

NVML_CC_SYSTEM_ENVIRONMENT_PROD
Environment is production.

NVML_CC_SYSTEM_ENVIRONMENT_SIM
Environment is simulation.

NVML_CC_SYSTEM_ENVIRONMENT_UNAVAILABLE
Confidential Compute Environment values.
Environment is unavailable.

NVML_CC_SYSTEM_FEATURE_BMSAI_ENABLED
Feature is enabled for Baremetal Secure AI.

NVML_CC_SYSTEM_FEATURE_DISABLED
Confidential Compute Feature Status values.

Feature is disabled.

NVML_CC_SYSTEM_FEATURE_ENABLED
Feature is enabled.

NVML_CC_SYSTEM_GPUS_CC_BMSAI_CAPABLE
GPUs are Baremetal Secure AI capable.

NVML_CC_SYSTEM_GPUS_CC_CAPABLE
GPUs are confidential compute capable.

NVML_CC_SYSTEM_GPUS_CC_NOT_CAPABLE
Confidenial Compute GPU Capabilities values.

GPUs are not confidential compute capable.

NVML_CC_SYSTEM_MULTIGPU_NONE
Confidential Compute Multigpu mode values.

Multi-GPU mode is none.

NVML_CC_SYSTEM_MULTIGPU_NVLE
Multi-GPU mode is NVLE.

NVML_CC_SYSTEM_MULTIGPU_PROTECTED_PCIE
Multi-GPU mode is protected PCIe.

NVML_CC_SYSTEM_TDISP_STATE_DISABLED
Confidential Compute TDISP state values.

The GPUs are not running behind a TDI.

NVML_CC_SYSTEM_TDISP_STATE_NOT_READY
The TDIs are UNLOCKED, i.e. TDISP FW-only mode.

NVML_CC_SYSTEM_TDISP_STATE_READY
The TDIs completed the TSM handshake and reached RUN, i.e. TDISP secure mode.

NVML_GPU_ATTESTATION_CERT_CHAIN_SIZE
Size of the attestation certificate chain.

NVML_GPU_CERT_CHAIN_SIZE

GPU Certificate Details.

Size of the certificate chain.

nvmlConfComputeGetKeyRotationThresholdInfo_v1

Version macro for *nvmlConfComputeGetKeyRotationThresholdInfo_v1_t*.

nvmlConfComputeSetKeyRotationThresholdInfo_v1

Version macro for *nvmlConfComputeSetKeyRotationThresholdInfo_v1_t*.

nvmlSystemConfComputeSettings_v1

Version macro for nvmlSystemConfComputeSettings_v1_t.

nvmlSystemConfComputeSettings_v2

Version macro for nvmlSystemConfComputeSettings_v2_t.

5.2.1.3.5 Typedefs

typedef *nvmlConfComputeGetKeyRotationThresholdInfo_v1_t*
nvmlConfComputeGetKeyRotationThresholdInfo_t

typedef *nvmlConfComputeSetKeyRotationThresholdInfo_v1_t*
nvmlConfComputeSetKeyRotationThresholdInfo_t

typedef *nvmlSystemConfComputeSettings_v1_t* **nvmlSystemConfComputeSettings_t**

5.2.1.4 Constants

5.2.1.4.1 Macros

NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlDeviceGetInforomVersion* and *nvmlDeviceGetInforomImageVersion* .

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

NVML_DEVICE_NAME_V2_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlDeviceGetName* .

NVML_DEVICE_PART_NUMBER_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlDeviceGetBoardPartNumber* .

NVML_DEVICE_SERIAL_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlDeviceGetSerial* .

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

NVML_DEVICE_UUID_V2_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlDeviceGetUUID* .

NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlDeviceGetVbiosVersion* .

NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlSystemGetDriverVersion* .

NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlSystemGetNVMLVersion* .

5.2.1.4.2 Macros

NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlDeviceGetInforomVersion* and *nvmlDeviceGetInforomImageVersion*.

NVML_DEVICE_NAME_BUFFER_SIZE

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

NVML_DEVICE_NAME_V2_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlDeviceGetName*.

NVML_DEVICE_PART_NUMBER_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlDeviceGetBoardPartNumber*.

NVML_DEVICE_SERIAL_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlDeviceGetSerial*.

NVML_DEVICE_UUID_BUFFER_SIZE

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

NVML_DEVICE_UUID_V2_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlDeviceGetUUID*.

NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlDeviceGetVbiosVersion*.

NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlSystemGetDriverVersion*.

NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlSystemGetNVMLVersion*.

5.2.1.5 Device Commands

This chapter describes NVML operations that change the state of the device.

Each of these requires root/admin access. Non-admin users will see an NVML_ERROR_NO_PERMISSION error code when invoking any of these methods.

5.2.1.5.1 Enumerations

nvmlClockLimitId_t

5.2.1.5.2 Functions

nvmlReturn_t *nvmlDeviceClearEccErrorCounts*(nvmlDevice_t device, nvmlEccCounterType_t counterType)
Clear the ECC error and other memory error counts for the device.

nvmlReturn_t *nvmlDeviceResetApplicationsClocks*(nvmlDevice_t device)

nvmlReturn_t *nvmlDeviceResetGpuLockedClocks*(nvmlDevice_t device)
Resets the gpu clock to the default value.

nvmlReturn_t *nvmlDeviceResetMemoryLockedClocks*(nvmlDevice_t device)
Resets the memory clock to the default value.

nvmlReturn_t *nvmlDeviceSetAPIRestriction*(nvmlDevice_t device, nvmlRestrictedAPI_t apiType, nvmlEnableState_t isRestricted)
Changes the root/admin restructions on certain APIs.

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

nvmlReturn_t *nvmlDeviceSetAutoBoostedClocksEnabled*(nvmlDevice_t device, nvmlEnableState_t enabled)
Try to set the current state of Auto Boosted clocks on a device.

nvmlReturn_t *nvmlDeviceSetComputeMode*(nvmlDevice_t device, nvmlComputeMode_t mode)
Set the compute mode for the device or MIG device.

nvmlReturn_t *nvmlDeviceSetDefaultAutoBoostedClocksEnabled*(nvmlDevice_t device, nvmlEnableState_t enabled, unsigned int flags)
Try to set the default state of Auto Boosted clocks on a device.

nvmlReturn_t *nvmlDeviceSetDefaultFanSpeed_v2*(nvmlDevice_t device, unsigned int fan)
Sets the speed of the fan control policy to default.

nvmlReturn_t *nvmlDeviceSetDriverModel*(nvmlDevice_t device, nvmlDriverModel_t driverModel, unsigned int flags)
Set the driver model for the device.

nvmlReturn_t *nvmlDeviceSetEccMode*(nvmlDevice_t device, nvmlEnableState_t ecc)
Set the ECC mode for the device.

nvmlReturn_t *nvmlDeviceSetFanControlPolicy*(nvmlDevice_t device, unsigned int fan, nvmlFanControlPolicy_t policy)
Sets current fan control policy.

nvmlReturn_t *nvmlDeviceSetFanSpeed_v2*(nvmlDevice_t device, unsigned int fan, unsigned int speed)
Sets the speed of a specified fan.

nvmlReturn_t *nvmlDeviceSetGpcClkVfOffset*(nvmlDevice_t device, int offset)

nvmlReturn_t *nvmlDeviceSetGpuLockedClocks*(nvmlDevice_t device, unsigned int minGpuClockMHz, unsigned int maxGpuClockMHz)
Set clocks that device will lock to.

nvmlReturn_t *nvmlDeviceSetGpuOperationMode*(nvmlDevice_t device, nvmlGpuOperationMode_t mode)
Sets new GOM.

nvmlReturn_t *nvmlDeviceSetMemClkVfOffset*(nvmlDevice_t device, int offset)

nvmlReturn_t *nvmlDeviceSetMemoryLockedClocks*(nvmlDevice_t device, unsigned int minMemClockMHz, unsigned int maxMemClockMHz)
Set memory clocks that device will lock to.

nvmlReturn_t *nvmlDeviceSetPersistenceMode*(nvmlDevice_t device, nvmlEnableState_t mode)
Set the persistence mode for the device.

nvmlReturn_t *nvmlDeviceSetPowerManagementLimit*(nvmlDevice_t device, unsigned int limit)
Set new power limit of this device.

nvmlReturn_t *nvmlDeviceSetRusdSettings_v1*(nvmlDevice_t device, nvmlRusdSettings_v1_t *settings)
Set Read-only user shared data (RUSD) settings for GPU.

nvmlReturn_t *nvmlDeviceSetTemperatureThreshold*(nvmlDevice_t device, nvmlTemperatureThresholds_t thresholdType, int *temp)
Sets the temperature threshold for the GPU with the specified threshold type in degrees C.

5.2.1.5.3 Enumerations

enum **nvmlClockLimitId_t**

Values:

enumerator **NVML_CLOCK_LIMIT_ID_RANGE_START**

enumerator **NVML_CLOCK_LIMIT_ID_TDP**

enumerator **NVML_CLOCK_LIMIT_ID_UNLIMITED**

5.2.1.5.4 Functions

nvmlReturn_t **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.

See also

- *nvmlDeviceGetDetailedEccErrors()*
- *nvmlDeviceGetTotalEccErrors()*

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

nvmlReturn_t **nvmlDeviceResetApplicationsClocks**(*nvmlDevice_t* device)

Deprecated:

Applications clocks are deprecated and will be removed in CUDA 14.0.

Please use *nvmlDeviceResetMemoryLockedClocks* for Memory Clocks and *nvmlDeviceResetGpuLockedClocks* for Graphics Clocks.

nvmlReturn_t **nvmlDeviceResetGpuLockedClocks**(*nvmlDevice_t* device)

Resets the gpu clock to the default value.

This is the gpu clock that will be used after system reboot or driver reload. Default values are idle clocks.

For Volta or newer fully supported devices.

See also

nvmlDeviceSetGpuLockedClocks

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

nvmlReturn_t **nvmlDeviceResetMemoryLockedClocks**(*nvmlDevice_t* device)

Resets the memory clock to the default value.

This is the memory clock that will be used after system reboot or driver reload. Default values are idle clocks.

For Ampere or newer fully supported devices.

See also

nvmlDeviceSetMemoryLockedClocks

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

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

Changes the root/admin restructions on certain APIs.

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

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

See also

nvmlRestrictedAPI_t

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

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

Deprecated:

Applications clocks are deprecated and will be removed in CUDA 14.0.

Please use *nvmlDeviceSetMemoryLockedClocks* for Memory Clocks and *nvmlDeviceSetGpuLockedClocks* for Graphics Clocks.

nvmlReturn_t **nvmlDeviceSetAutoBoostedClocksEnabled**(
nvmlDevice_t device, *nvmlEnableState_t* enabled,
)

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

For Kepler or newer fully supported devices.

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

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

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

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

nvmlReturn_t **nvmlDeviceSetComputeMode**(
    nvmlDevice_t device, *nvmlComputeMode_t* mode,
)

Set the compute mode for the device or MIG 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.

See also

nvmlDeviceGetComputeMode()

Note

If MIG is enabled on a GPU, device must be MIG device handle.

Parameters

- **device** – The identifier of the target device handle or MIG device handle
- **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

nvmlReturn_t **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

nvmlReturn_t **nvmlDeviceSetDefaultFanSpeed_v2**(
    nvmlDevice_t device, unsigned int fan,
)

Sets the speed of the fan control policy to default.

For all cuda-capable discrete products with fans

Parameters

- **device** – The identifier of the target device
- **fan** – The index of the fan, starting at zero

Returns

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

nvmlReturn_t **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.

Driver-model availability is architecture-specific; attempting to set an unsupported driver model returns NVML_ERROR_NOT_SUPPORTED.

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

This operation takes effect after the next reboot.

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

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

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

See also

nvmlDeviceGetDriverModel()

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

nvmlReturn_t **nvmlDeviceSetEccMode**(
    nvmlDevice_t device, *nvmlEnableState_t* ecc,
)

Set the ECC mode for the device.

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

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

This operation takes effect after the next reboot.

See *nvmlEnableState_t* for details on available modes.

See also

nvmlDeviceGetEccMode()

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

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

Sets current fan control policy.

For Maxwell or newer fully supported devices.

Requires privileged user.

For all cuda-capable discrete products with fans

Parameters

- **device** – The identifier of the target *device*
- **fan** – The index of the fan, starting at zero
- **policy** – The fan control *policy* to set

Returns

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

nvmlReturn_t **nvmlDeviceSetFanSpeed_v2**(
    nvmlDevice_t device, unsigned int fan, unsigned int speed,
)

Sets the speed of a specified fan.

WARNING: This function changes the fan control policy to manual. It means that YOU have to monitor the temperature and adjust the fan speed accordingly. If you set the fan speed too low you can burn your GPU! Use nvmlDeviceSetDefaultFanSpeed_v2 to restore default control policy.

For all cuda-capable discrete products with fans that are Maxwell or Newer.

return

- *NVML_SUCCESS* if the fan speed has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if the device is not valid, or the speed is outside acceptable ranges, or if the fan index doesn't reference an actual fan.
- *NVML_ERROR_NOT_SUPPORTED* if the device is older than Maxwell.
- *NVML_ERROR_UNKNOWN* if there was an unexpected error.

Parameters

- **device** – The identifier of the target device
- **fan** – The index of the fan, starting at zero
- **speed** – The target speed of the fan [0-100] in % of max speed

nvmlReturn_t **nvmlDeviceSetGpcClkVfOffset**(
nvmlDevice_t device, int offset,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlDeviceSetClockOffsets* instead. It works on Maxwell onwards GPU architectures.

Set the GPCCLK VF offset value

Parameters

- **device** – **[in]** The identifier of the target device
- **offset** – **[in]** The GPCCLK VF offset value to set

Returns

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

nvmlReturn_t **nvmlDeviceSetGpuLockedClocks**(
nvmlDevice_t device, unsigned int minGpuClockMHz, unsigned int maxGpuClockMHz,
)

Set clocks that device will lock to.

Sets the clocks that the device will be running at to the value in the range of minGpuClockMHz to maxGpuClockMHz.

Can be used as a setting to request constant performance.

This can be called with a pair of integer clock frequencies in MHz, or a pair of /ref nvmlClockLimitId_t values. See the table below for valid combinations of these values.

minGpuClock | maxGpuClock | Effect ————+———-—+————————————— tdp | tdp | Lock clock to TDP unlimited | tdp | Upper bound is TDP but clock may drift below this tdp | unlimited | Lower bound is TDP but clock may boost above this unlimited | unlimited | Unlocked (== nvmlDeviceResetGpuLockedClocks)

If one arg takes one of these values, the other must be one of these values as well. Mixed numeric and symbolic calls return NVML_ERROR_INVALID_ARGUMENT.

Requires root/admin permissions.

After system reboot or driver reload GPU clocks go back to their default value. See *nvmlDeviceResetGpuLockedClocks*.

For Volta or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **minGpuClockMHz** – Requested minimum gpu clock in MHz
- **maxGpuClockMHz** – Requested maximum gpu clock in MHz

Returns

- *NVML_SUCCESS* if new settings were successfully set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *minGpuClockMHz* and *maxGpuClockMHz* is not a valid clock combination
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_NOT_SUPPORTED* if the device doesn't support this feature
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

Sets new GOM.

See *nvmlGpuOperationMode_t* for details.

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

Changing GOMs requires a reboot. The reboot requirement might be removed in the future.

Compute only GOMs don't support graphics acceleration. Under windows switching to these GOMs when pending driver model is WDDM is not supported. See *nvmlDeviceSetDriverModel*.

See also

nvmlGpuOperationMode_t

See also

nvmlDeviceGetGpuOperationMode

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

nvmlReturn_t **nvmlDeviceSetMemClkVfOffset**(
    nvmlDevice_t device, int offset,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlDeviceSetClockOffsets* instead. It works on Maxwell onwards GPU architectures.

Set the MemClk (Memory Clock) VF offset value. It requires elevated privileges.

Parameters

- **device** – **[in]** The identifier of the target device
- **offset** – **[in]** The MemClk VF offset value to set

Returns

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

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

nvmlReturn_t **nvmlDeviceSetMemoryLockedClocks**(
nvmlDevice_t device, unsigned int minMemClockMHz, unsigned int maxMemClockMHz,
)

Set memory clocks that device will lock to.

Sets the device's memory clocks to the value in the range of minMemClockMHz to maxMemClockMHz.

Can be used as a setting to request constant performance.

Requires root/admin permissions.

After system reboot or driver reload memory clocks go back to their default value. See *nvmlDeviceResetMemoryLockedClocks*.

For Ampere or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **minMemClockMHz** – Requested minimum memory clock in MHz
- **maxMemClockMHz** – Requested maximum memory clock in MHz

Returns

- *NVML_SUCCESS* if new settings were successfully set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *minGpuClockMHz* and *maxGpuClockMHz* is not a valid clock combination
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_NOT_SUPPORTED* if the device doesn't support this feature
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **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.

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

See also

nvmlDeviceGetPersistenceMode()

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

nvmlReturn_t **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.

See also

nvmlDeviceGetPowerManagementLimitConstraints

See also

nvmlDeviceGetPowerManagementDefaultLimit

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

nvmlReturn_t **nvmlDeviceSetRusdSettings_v1**(
    nvmlDevice_t device, *nvmlRusdSettings_v1_t* *settings,
)

Set Read-only user shared data (RUSD) settings for GPU.

Requires root/admin permissions.

Parameters

- **device** – The identifier of the target device
- **settings** – Reference to *nvmlRusdSettings_v1_t* struct

Returns

- *NVML_SUCCESS* if the RUSD setting was successfully set
- *NVML_ERROR_INVALID_ARGUMENT* if device is invalid or state is NULL
- *NVML_ERROR_NO_PERMISSION* if user does not have permission to change feature state
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by NVIDIA kernel driver

- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the input version is not supported

nvmlReturn_t **nvmlDeviceSetTemperatureThreshold**(
nvmlDevice_t device, *nvmlTemperatureThresholds_t* thresholdType, int *temp,
)

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

For Maxwell or newer fully supported devices.

See *nvmlTemperatureThresholds_t* for details on available temperature thresholds.

Parameters

- **device** – The identifier of the target device
- **thresholdType** – The type of threshold value to be set
- **temp** – Reference which hold the value to be set

Returns

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

5.2.1.6 Device Enums

5.2.1.6.1 Macros

MAX_CLK_DOMAINS
Max Clock Monitors available.

NVML_ADAPTIVE_CLOCKING_INFO_STATUS_DISABLED
Adaptive clocking status.

NVML_ADAPTIVE_CLOCKING_INFO_STATUS_ENABLED
Adaptive clocking is enabled.

NVML_BUS_TYPE_AGP
AGP bus.

NVML_BUS_TYPE_FPCI
FPCI bus.

NVML_BUS_TYPE_PCI
PCI bus.

NVML_BUS_TYPE_PCIE
PCI-Express bus.

NVML_BUS_TYPE_UNKNOWN
PCI bus types.

NVML_DEVICE_ARCH_ADA
Devices based on the NVIDIA Ada architecture.

NVML_DEVICE_ARCH_AMPERE
Devices based on the NVIDIA Ampere architecture.

NVML_DEVICE_ARCH_BLACKWELL
Devices based on the NVIDIA Blackwell architecture.

NVML_DEVICE_ARCH_DLA
Devices based on the NVIDIA DLA architecture.

NVML_DEVICE_ARCH_DLA2
Devices based on the NVIDIA DLA2 architecture.

NVML_DEVICE_ARCH_HOPPER
Devices based on the NVIDIA Hopper architecture.

NVML_DEVICE_ARCH_KEPLER
Simplified chip architecture.

NVML_DEVICE_ARCH_MAXWELL
Devices based on the NVIDIA Maxwell architecture.

NVML_DEVICE_ARCH_NPU3
Devices based on the NVIDIA NPU3 architecture.

NVML_DEVICE_ARCH_PASCAL
Devices based on the NVIDIA Pascal architecture.

NVML_DEVICE_ARCH_RUBIN
Devices based on the NVIDIA Rubin architecture.

NVML_DEVICE_ARCH_TURING
Devices based on the NVIDIA Turing architecture.

NVML_DEVICE_ARCH_UNKNOWN
Anything else, presumably something newer.

NVML_DEVICE_ARCH_VOLTA
Devices based on the NVIDIA Volta architecture.

NVML_DEVICE_HOSTNAME_BUFFER_SIZE
Buffer size for hostname string.

NVML_DOUBLE_BIT_ECC
Double bit ECC errors.

NVML_FAN_POLICY_MANUAL
Manual fan control policy.

NVML_FAN_POLICY_TEMPERATURE_CONTINOUS_SW
Device Power Modes.

NVML_GSP_FIRMWARE_VERSION_BUF_SIZE
GSP firmware.

NVML_MAX_GPU_PERF_PSTATES
Maximum number of GPU performance states.

NVML_MAX_GPU_UTILIZATIONS
Maximum number of GPU utilization domains.

NVML_PCIE_ATOMICS_CAP_CAS128
128-bit compare and swap.

NVML_PCIE_ATOMICS_CAP_CAS32
32-bit compare and swap.

NVML_PCIE_ATOMICS_CAP_CAS64
64-bit compare and swap.

NVML_PCIE_ATOMICS_CAP_FETCHADD32
32-bit fetch and add.

NVML_PCIE_ATOMICS_CAP_FETCHADD64
64-bit fetch and add.

NVML_PCIE_ATOMICS_CAP_SWAP32
32-bit swap.

NVML_PCIE_ATOMICS_CAP_SWAP64
64-bit swap.

NVML_PCIE_ATOMICS_OPS_MAX
Maximum number of PCIe atomics operations.

NVML_PCIE_LINK_MAX_SPEED_16000MBPS
16000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_2500MBPS
2500 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_32000MBPS
32000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_5000MBPS
5000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_64000MBPS
64000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_8000MBPS
8000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_INVALID
Device PCIE link Max Speed.

NVML_PERF_MODES_BUFFER_SIZE
Buffer size for performance modes strings.

NVML_POWER_MIZER_MODE_ADAPTIVE
Device powerMizer modes.

NVML_POWER_MIZER_MODE_AUTO
PowerMizer mode is driver controlled.

NVML_POWER_MIZER_MODE_PREFER_CONSISTENT_PERFORMANCE
lock to GPU base clocks

NVML_POWER_MIZER_MODE_PREFER_MAXIMUM_PERFORMANCE
raise GPU clocks to favor maximum performance, to the extent that thermal and other constraints allow

NVML_POWER_SCOPE_COUNT
Maximum number of power scopes.

NVML_POWER_SCOPE_GPU
Device Scope - This is useful to retrieve the telemetry at GPU and module (e.g.

NVML_POWER_SCOPE_GPU_BASE
Targets base GPU power.

NVML_POWER_SCOPE_MEMORY
Targets the GPU Memory.

NVML_POWER_SCOPE_MODULE
Targets the whole module.

NVML_POWER_SOURCE_AC
Device Power Source.

NVML_POWER_SOURCE_BATTERY
Battery power source.

NVML_POWER_SOURCE_UNDERSIZED
Undersized power source.

NVML_RUSD_POLL_ALL
Enable RUSD polling on all groups.

NVML_RUSD_POLL_CLOCK
Enable RUSD polling on clock group.

NVML_RUSD_POLL_FAN
Enable RUSD polling on fan group.

NVML_RUSD_POLL_MEMORY
Enable RUSD polling on memory group.

NVML_RUSD_POLL_NONE
Disable RUSD polling on all metric groups.

NVML_RUSD_POLL_PCI
Enable RUSD polling on pci group.

NVML_RUSD_POLL_PERF
Enable RUSD polling on performance group.

NVML_RUSD_POLL_POWER
Enable RUSD polling on power group.

NVML_RUSD_POLL_PROC_UTIL
Enable RUSD polling on process utilization group.

NVML_RUSD_POLL_THERMAL
Enable RUSD polling on thermal group.

NVML_SINGLE_BIT_ECC
Single bit ECC errors.

nvmlClockOffset_v1
Version macro for *nvmlClockOffset_v1_t* .

nvmlDeviceCurrentClockFreqs_v1
Version macro for *nvmlDeviceCurrentClockFreqs_v1_t* .

nvmlDevicePerfModes_v1
Version macro for *nvmlDevicePerfModes_v1_t* .

nvmlDramEncryptionInfo_v1
Version macro for *nvmlDramEncryptionInfo_v1_t* .

nvmlEccBitType_t
ECC bit types.

nvmlEccSramErrorStatus_v1
Version macro for *nvmlEccSramErrorStatus_v1_t* .

nvmlEccSramUniqueUncorrectedErrorCounts_v1
Version macro for *nvmlEccSramUniqueUncorrectedErrorCounts_v1_t* .

nvmlFanSpeedInfo_v1
Version macro for *nvmlFanSpeedInfo_v1_t* .

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

nvmlFlagForce
Force flag.

nvmlMarginTemperature_v1
Version macro for *nvmlMarginTemperature_v1_t* .

nvmlPlatformInfo_v1
Version macro for *nvmlPlatformInfo_v1_t* .

nvmlPlatformInfo_v2
Version macro for *nvmlPlatformInfo_v2_t* .

nvmlPowerValue_v2
Version macro for *nvmlPowerValue_v2_t* .

nvmlProcessesUtilizationInfo_v1

nvmlRusdSettings_v1
Version macro for *nvmlRusdSettings_v1_t* .

5.2.1.6.2 Enumerations

nvmlBrandType_t

nvmlClockId_t
Clock Ids.

nvmlClockType_t
Clock types.

nvmlComputeMode_t
Compute mode.

nvmlDriverModel_t
Driver models.

nvmlEccCounterType_t
ECC counter types.

nvmlEnableState_t
Generic enable/disable enum.

nvmlGpuOperationMode_t
GPU Operation Mode.

nvmlGpuUtilizationDomainId_t
Represents the GPU utilization domains.

nvmlInforomObject_t
Available infoROM objects.

nvmlMemoryErrorType_t
Memory error types.

nvmlMemoryLocation_t
See *nvmlDeviceGetMemoryErrorCounter* .

nvmlPageRetirementCause_t
Causes for page retirement.

nvmlPstates_t
Allowed PStates.

nvmlRestrictedAPI_t
API types that allow changes to default permission restrictions.

nvmlReturn_t
Return values for NVML API calls.

nvmlTemperatureSensors_t
Temperature sensors.

nvmlTemperatureThresholds_t
Temperature thresholds.

5.2.1.6.3 Structs

nvmlAdaptiveTgpModeInfo_v1_t
Stores Adaptive TGP Mode state and telemetry for a GPU.

nvmlClkMonFaultInfo_t
Clock Monitor error types.

nvmlClkMonStatus_t
Clock Monitor Status.

nvmlClockOffset_v1_t
Clock offset info.

nvmlDeviceCurrentClockFreqs_v1_t
Device current clocks string.

nvmlDevicePerfModes_v1_t
Device performance modes string.

nvmlDevicePowerMizerModes_v1_t

nvmlDramEncryptionInfo_v1_t
DRAM Encryption Info.

nvmlEccSramErrorStatus_v1_t
Structure to store SRAM uncorrectable error counters.

nvmlEccSramUniqueUncorrectedErrorCounts_v1_t

nvmlEccSramUniqueUncorrectedErrorEntry_v1_t

nvmlFanSpeedInfo_v1_t
Fan speed info.

nvmlGpuDynamicPstatesInfo_t

nvmlHostname_v1_t
Stores the hostname string assigned to a GPU device.

nvmlMarginTemperature_v1_t
Margin temperature values.

nvmlPlatformInfo_v1_t
Structure to store platform information.

nvmlPlatformInfo_v2_t
Structure to store platform information (v2)

nvmlPowerValue_v2_t
Contains the power management limit.

nvmlProcessUtilizationInfo_v1_t
Structure to store utilization value and process Id version 1.

nvmlProcessUtilizationSample_t
Structure to store utilization value and process Id.

nvmlProcessesUtilizationInfo_v1_t
Structure to store utilization and process ID for each running process version 1.

nvmlRemappedRowsInfo_v2_t

nvmlRusdSettings_v1_t
Stores RUSD polling configuration for a GPU.

5.2.1.6.4 Typedefs

nvmlBusType_t

nvmlClockOffset_t

nvmlDeviceArchitecture_t

nvmlDeviceCurrentClockFreqs_t

nvmlDevicePerfModes_t

nvmlDramEncryptionInfo_t

nvmlEccSramErrorStatus_t

nvmlEccSramUniqueUncorrectedErrorCounts_t

nvmlFanControlPolicy_t

nvmlFanSpeedInfo_t

nvmlMarginTemperature_t

nvmlPlatformInfo_t

nvmlPowerScopeType_t

nvmlPowerSource_t

nvmlProcessesUtilizationInfo_t

5.2.1.6.5 Macros

MAX_CLK_DOMAINS

Max Clock Monitors available.

Maximum number of clock domains.

NVML_ADAPTIVE_CLOCKING_INFO_STATUS_DISABLED

Adaptive clocking status.

Adaptive clocking is disabled.

NVML_ADAPTIVE_CLOCKING_INFO_STATUS_ENABLED

Adaptive clocking is enabled.

NVML_BUS_TYPE_AGP

AGP bus.

NVML_BUS_TYPE_FPCI

FPCI bus.

NVML_BUS_TYPE_PCI

PCI bus.

NVML_BUS_TYPE_PCIE

PCI-Express bus.

NVML_BUS_TYPE_UNKNOWN

PCI bus types.

Unknown bus type.

NVML_DEVICE_ARCH_ADA

Devices based on the NVIDIA Ada architecture.

NVML_DEVICE_ARCH_AMPERE

Devices based on the NVIDIA Ampere architecture.

NVML_DEVICE_ARCH_BLACKWELL

Devices based on the NVIDIA Blackwell architecture.

NVML_DEVICE_ARCH_DLA

Devices based on the NVIDIA DLA architecture.

NVML_DEVICE_ARCH_DLA2

Devices based on the NVIDIA DLA2 architecture.

NVML_DEVICE_ARCH_HOPPER

Devices based on the NVIDIA Hopper architecture.

NVML_DEVICE_ARCH_KEPLER

Simplified chip architecture.

Devices based on the NVIDIA Kepler architecture

NVML_DEVICE_ARCH_MAXWELL

Devices based on the NVIDIA Maxwell architecture.

NVML_DEVICE_ARCH_NPU3

Devices based on the NVIDIA NPU3 architecture.

NVML_DEVICE_ARCH_PASCAL

Devices based on the NVIDIA Pascal architecture.

NVML_DEVICE_ARCH_RUBIN

Devices based on the NVIDIA Rubin architecture.

NVML_DEVICE_ARCH_TURING

Devices based on the NVIDIA Turing architecture.

NVML_DEVICE_ARCH_UNKNOWN
Anything else, presumably something newer.

NVML_DEVICE_ARCH_VOLTA
Devices based on the NVIDIA Volta architecture.

NVML_DEVICE_HOSTNAME_BUFFER_SIZE
Buffer size for hostname string.

NVML_DOUBLE_BIT_ECC
Double bit ECC errors.

Deprecated:
Mapped to *NVML_MEMORY_ERROR_TYPE_UNCORRECTED*

Deprecated double bit ECC error. See *NVML_MEMORY_ERROR_TYPE_UNCORRECTED*.

NVML_FAN_POLICY_MANUAL
Manual fan control policy.

NVML_FAN_POLICY_TEMPERATURE_CONTINOUS_SW
Device Power Modes.

Device Fan control policy Temperature-controlled fan policy.

NVML_GSP_FIRMWARE_VERSION_BUF_SIZE
GSP firmware.

Buffer size for GSP firmware version string.

NVML_MAX_GPU_PERF_PSTATES
Maximum number of GPU performance states.

NVML_MAX_GPU_UTILIZATIONS
Maximum number of GPU utilization domains.

NVML_PCIE_ATOMICS_CAP_CAS128
128-bit compare and swap.

NVML_PCIE_ATOMICS_CAP_CAS32
32-bit compare and swap.

NVML_PCIE_ATOMICS_CAP_CAS64
64-bit compare and swap.

NVML_PCIE_ATOMICS_CAP_FETCHADD32
32-bit fetch and add.

NVML_PCIE_ATOMICS_CAP_FETCHADD64
64-bit fetch and add.

NVML_PCIE_ATOMICS_CAP_SWAP32
32-bit swap.

NVML_PCIE_ATOMICS_CAP_SWAP64
64-bit swap.

NVML_PCIE_ATOMICS_OPS_MAX
Maximum number of PCIe atomics operations.

NVML_PCIE_LINK_MAX_SPEED_16000MBPS
16000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_2500MBPS
2500 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_32000MBPS
32000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_5000MBPS
5000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_64000MBPS
64000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_8000MBPS
8000 MB/s PCIe link speed.

NVML_PCIE_LINK_MAX_SPEED_INVALID
Device PCIE link Max Speed.
Invalid PCIe link speed.

NVML_PERF_MODES_BUFFER_SIZE
Buffer size for performance modes strings.

NVML_POWER_MIZER_MODE_ADAPTIVE
Device powerMizer modes.
adjust GPU clocks based on GPU utilization

NVML_POWER_MIZER_MODE_AUTO
PowerMizer mode is driver controlled.

NVML_POWER_MIZER_MODE_PREFER_CONSISTENT_PERFORMANCE

lock to GPU base clocks

NVML_POWER_MIZER_MODE_PREFER_MAXIMUM_PERFORMANCE

raise GPU clocks to favor maximum performance, to the extent that thermal and other constraints allow

NVML_POWER_SCOPE_COUNT

Maximum number of power scopes.

NVML_POWER_SCOPE_GPU

Device Scope - This is useful to retrieve the telemetry at GPU and module (e.g.

GPU + CPU) level Targets only GPU

NVML_POWER_SCOPE_GPU_BASE

Targets base GPU power.

NVML_POWER_SCOPE_MEMORY

Targets the GPU Memory.

NVML_POWER_SCOPE_MODULE

Targets the whole module.

NVML_POWER_SOURCE_AC

Device Power Source.

AC power source.

NVML_POWER_SOURCE_BATTERY

Battery power source.

NVML_POWER_SOURCE_UNDERSIZED

Undersized power source.

NVML_RUSD_POLL_ALL

Enable RUSD polling on all groups.

NVML_RUSD_POLL_CLOCK

Enable RUSD polling on clock group.

NVML_RUSD_POLL_FAN

Enable RUSD polling on fan group.

NVML_RUSD_POLL_MEMORY

Enable RUSD polling on memory group.

NVML_RUSD_POLL_NONE
Disable RUSD polling on all metric groups.

NVML_RUSD_POLL_PCI
Enable RUSD polling on pci group.

NVML_RUSD_POLL_PERF
Enable RUSD polling on performance group.

NVML_RUSD_POLL_POWER
Enable RUSD polling on power group.

NVML_RUSD_POLL_PROC_UTIL
Enable RUSD polling on process utilization group.

NVML_RUSD_POLL_THERMAL
Enable RUSD polling on thermal group.

NVML_SINGLE_BIT_ECC
Single bit ECC errors.

Deprecated:
Mapped to *NVML_MEMORY_ERROR_TYPE_CORRECTED*

Deprecated single bit ECC error. See *NVML_MEMORY_ERROR_TYPE_CORRECTED*.

nvmlClockOffset_v1
Version macro for nvmlClockOffset_v1_t.

nvmlDeviceCurrentClockFreqs_v1
Version macro for nvmlDeviceCurrentClockFreqs_v1_t.

nvmlDevicePerfModes_v1
Version macro for nvmlDevicePerfModes_v1_t.

nvmlDramEncryptionInfo_v1
Version macro for nvmlDramEncryptionInfo_v1_t.

nvmlEccBitType_t
ECC bit types.

Deprecated:
See *nvmlMemoryErrorType_t* for a more flexible type

Deprecated ECC bit type. See *nvmlMemoryErrorType_t*.

nvmlEccSramErrorStatus_v1

Version macro for nvmlEccSramErrorStatus_v1_t.

nvmlEccSramUniqueUncorrectedErrorCounts_v1

Version macro for *nvmlEccSramUniqueUncorrectedErrorCounts_v1_t*.

nvmlFanSpeedInfo_v1

Version macro for nvmlFanSpeedInfo_v1_t.

nvmlFlagDefault

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

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

nvmlFlagForce

Force flag.

nvmlMarginTemperature_v1

Version macro for nvmlMarginTemperature_v1_t.

nvmlPlatformInfo_v1

Version macro for nvmlPlatformInfo_v1_t.

nvmlPlatformInfo_v2

Version macro for nvmlPlatformInfo_v2_t.

nvmlPowerValue_v2

Version macro for nvmlPowerValue_v2_t.

nvmlProcessesUtilizationInfo_v1

nvmlRusdSettings_v1

Version macro for nvmlRusdSettings_v1_t.

5.2.1.6.6 Enumerations

enum **nvmlBrandType_t**

- The Brand of the GPU

Values:

enumerator **NVML_BRAND_UNKNOWN**

enumerator **NVML_BRAND_QUADRO**

enumerator **NVML_BRAND_TESLA**

enumerator **NVML_BRAND_NVS**

enumerator **NVML_BRAND_GRID**

enumerator **NVML_BRAND_GEFORCE**

enumerator **NVML_BRAND_TITAN**

enumerator **NVML_BRAND_NVIDIA_VAPPS**

enumerator **NVML_BRAND_NVIDIA_VPC**

enumerator **NVML_BRAND_NVIDIA_VCS**

enumerator **NVML_BRAND_NVIDIA_VWS**

enumerator **NVML_BRAND_NVIDIA_CLOUD_GAMING**

enumerator **NVML_BRAND_NVIDIA_VGAMING**

enumerator **NVML_BRAND_QUADRO_RTX**

enumerator **NVML_BRAND_NVIDIA_RTX**

enumerator **NVML_BRAND_NVIDIA**

enumerator **NVML_BRAND_GEFORCE_RTX**

enumerator **NVML_BRAND_TITAN_RTX**

enumerator **NVML_BRAND_NVIDIA_DLA**

enumerator **NVML_BRAND_NVIDIA_NPU**

enumerator **NVML_BRAND_NVIDIA_VGAMEDEV**

enumerator **NVML_BRAND_COUNT**

enum **nvmlClockId_t**

Clock Ids.

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

Values:

enumerator **NVML_CLOCK_ID_CURRENT**

Current actual clock value.

enumerator **NVML_CLOCK_ID_APP_CLOCK_TARGET**

Target application clock. Deprecated, do not use.

enumerator **NVML_CLOCK_ID_APP_CLOCK_DEFAULT**

Default application clock target Deprecated, do not use.

enumerator **NVML_CLOCK_ID_CUSTOMER_BOOST_MAX**

OEM-defined maximum clock rate.

enumerator **NVML_CLOCK_ID_COUNT**

Count of Clock Ids.

enum **nvmlClockType_t**

Clock types.

All speeds are in Mhz.

Values:

enumerator **NVML_CLOCK_GRAPHICS**

Graphics clock domain.

enumerator **NVML_CLOCK_SM**

SM clock domain.

enumerator **NVML_CLOCK_MEM**

Memory clock domain.

enumerator **NVML_CLOCK_VIDEO**

Video encoder/decoder clock domain.

enumerator **NVML_CLOCK_COUNT**

Count of clock types.

enum **nvmlComputeMode_t**

Compute mode.

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

Values:

enumerator **NVML_COMPUTEMODE_DEFAULT**

Default compute mode — multiple contexts per device.

enumerator **NVML_COMPUTEMODE_EXCLUSIVE_THREAD**

Support Removed.

enumerator **NVML_COMPUTEMODE_PROHIBITED**

Compute-prohibited mode — no contexts per device.

enumerator **NVML_COMPUTEMODE_EXCLUSIVE_PROCESS**

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

enumerator **NVML_COMPUTEMODE_COUNT**

enum **nvmlDriverModel_t**

Driver models.

Windows only.

Values:

enumerator **NVML_DRIVER_WDDM**

WDDM driver model — GPU treated as a display device.

enumerator **NVML_DRIVER_WDM**

WDM (TCC) model (deprecated) — GPU treated as a generic compute device.

enumerator **NVML_DRIVER_MCDM**

MCDM driver model — GPU treated as a Microsoft compute device.

enum **nvmlEccCounterType_t**

ECC counter types.

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

Values:

enumerator **NVML_VOLATILE_ECC**

Volatile counts are reset each time the driver loads.

enumerator **NVML_AGGREGATE_ECC**

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

enumerator **NVML_ECC_COUNTER_TYPE_COUNT**

Count of memory counter types.

enum **nvmlEnableState_t**

Generic enable/disable enum.

Values:

enumerator **NVML_FEATURE_DISABLED**

Feature disabled.

enumerator **NVML_FEATURE_ENABLED**

Feature enabled.

enum **nvmlGpuOperationMode_t**

GPU Operation Mode.

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

Each GOM is designed to meet specific user needs.

Values:

enumerator **NVML_GOM_ALL_ON**

Everything is enabled and running at full speed.

enumerator **NVML_GOM_COMPUTE**

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

enumerator **NVML_GOM_LOW_DP**

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

enum **nvmlGpuUtilizationDomainId_t**

Represents the GPU utilization domains.

Values:

enumerator **NVML_GPU_UTILIZATION_DOMAIN_GPU**

Graphics engine domain.

enumerator **NVML_GPU_UTILIZATION_DOMAIN_FB**

Frame buffer domain.

enumerator **NVML_GPU_UTILIZATION_DOMAIN_VID**

Video engine domain.

enumerator **NVML_GPU_UTILIZATION_DOMAIN_BUS**

Bus interface domain.

enum **nvmlInforomObject_t**

Available infoROM objects.

Values:

enumerator **NVML_INFOROM_OEM**

An object defined by OEM.

enumerator **NVML_INFOROM_ECC**

The ECC object determining the level of ECC support.

enumerator **NVML_INFOROM_POWER**

The power management object.

enumerator **NVML_INFOROM_DEN**

DRAM Encryption object.

enumerator **NVML_INFOROM_COUNT**

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

enum **nvmlMemoryErrorType_t**

Memory error types.

Values:

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

enumerator **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

enumerator **NVML_MEMORY_ERROR_TYPE_COUNT**

Count of memory error types.

enum **nvmlMemoryLocation_t**

See *nvmlDeviceGetMemoryErrorCounter*.

Values:

enumerator **NVML_MEMORY_LOCATION_L1_CACHE**

GPU L1 Cache.

enumerator **NVML_MEMORY_LOCATION_L2_CACHE**
GPU L2 Cache.

enumerator **NVML_MEMORY_LOCATION_DRAM**
Turing+ DRAM.

enumerator **NVML_MEMORY_LOCATION_DEVICE_MEMORY**
GPU Device Memory.

enumerator **NVML_MEMORY_LOCATION_REGISTER_FILE**
GPU Register File.

enumerator **NVML_MEMORY_LOCATION_TEXTURE_MEMORY**
GPU Texture Memory.

enumerator **NVML_MEMORY_LOCATION_TEXTURE_SHM**
Shared memory.

enumerator **NVML_MEMORY_LOCATION_CBU**
CBU.

enumerator **NVML_MEMORY_LOCATION_SRAM**
Turing+ SRAM.

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

enum **nvmlPageRetirementCause_t**
Causes for page retirement.
Values:

enumerator **NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS**
Page was retired due to multiple single bit ECC error.

enumerator **NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR**
Page was retired due to double bit ECC error.

enumerator **NVML_PAGE_RETIREMENT_CAUSE_COUNT**

enum **nvmlPstates_t**
Allowed PStates.
Values:

enumerator **NVML_PSTATE_0**
Performance state 0 — Maximum Performance.

enumerator **NVML_PSTATE_1**
Performance state 1.

enumerator **NVML_PSTATE_2**
Performance state 2.

enumerator **NVML_PSTATE_3**
Performance state 3.

enumerator **NVML_PSTATE_4**
Performance state 4.

enumerator **NVML_PSTATE_5**
Performance state 5.

enumerator **NVML_PSTATE_6**
Performance state 6.

enumerator **NVML_PSTATE_7**
Performance state 7.

enumerator **NVML_PSTATE_8**
Performance state 8.

enumerator **NVML_PSTATE_9**
Performance state 9.

enumerator **NVML_PSTATE_10**
Performance state 10.

enumerator **NVML_PSTATE_11**
Performance state 11.

enumerator **NVML_PSTATE_12**
Performance state 12.

enumerator **NVML_PSTATE_13**
Performance state 13.

enumerator **NVML_PSTATE_14**
Performance state 14.

enumerator **NVML_PSTATE_15**
Performance state 15 — Minimum Performance.

enumerator **NVML_PSTATE_UNKNOWN**

Unknown performance state.

enum **nvmlRestrictedAPI_t**

API types that allow changes to default permission restrictions.

Values:

enumerator **NVML_RESTRICTED_API_SET_APPLICATION_CLOCKS**

APIs that change application clocks, see nvmlDeviceSetApplicationsClocks and see nvmlDeviceResetApplicationsClocks. Deprecated, keeping definition for backward compatibility.

enumerator **NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS**

APIs that enable/disable Auto Boosted clocks see nvmlDeviceSetAutoBoostedClocksEnabled.

enumerator **NVML_RESTRICTED_API_COUNT**

enum **nvmlReturn_t**

Return values for NVML API calls.

Values:

enumerator **NVML_SUCCESS**

The operation was successful.

enumerator **NVML_ERROR_UNINITIALIZED**

NVML was not first initialized with nvmlInit()

enumerator **NVML_ERROR_INVALID_ARGUMENT**

A supplied argument is invalid.

enumerator **NVML_ERROR_NOT_SUPPORTED**

The requested operation is not available on target device.

enumerator **NVML_ERROR_NO_PERMISSION**

The current user does not have permission for operation.

enumerator **NVML_ERROR_ALREADY_INITIALIZED**

Deprecated: Multiple initializations are now allowed through ref counting.

enumerator **NVML_ERROR_NOT_FOUND**

A query to find an object was unsuccessful.

enumerator **NVML_ERROR_INSUFFICIENT_SIZE**

An input argument is not large enough.

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

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

enumerator **NVML_ERROR_TIMEOUT**
User provided timeout passed.

enumerator **NVML_ERROR_IRQ_ISSUE**
NVIDIA Kernel detected an interrupt issue with a GPU.

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

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

enumerator **NVML_ERROR_CORRUPTED_INFOROM**
infoROM is corrupted

enumerator **NVML_ERROR_GPU_IS_LOST**
The GPU has fallen off the bus or has otherwise become inaccessible.

enumerator **NVML_ERROR_RESET_REQUIRED**
The GPU requires a reset before it can be used again.

enumerator **NVML_ERROR_OPERATING_SYSTEM**
A generic operating system error occurred.

enumerator **NVML_ERROR_LIB_RM_VERSION_MISMATCH**
RM detects a driver/library version mismatch.

enumerator **NVML_ERROR_IN_USE**
An operation cannot be performed because the GPU is currently in use.

enumerator **NVML_ERROR_MEMORY**
Insufficient memory.

enumerator **NVML_ERROR_NO_DATA**
No data.

enumerator **NVML_ERROR_VGPU_ECC_NOT_SUPPORTED**
The requested vgpu operation is not available on target device, becasue ECC is enabled.

enumerator **NVML_ERROR_INSUFFICIENT_RESOURCES**
Ran out of critical resources, other than memory.

enumerator **NVML_ERROR_FREQ_NOT_SUPPORTED**
Ran out of critical resources, other than memory.

enumerator **NVML_ERROR_ARGUMENT_VERSION_MISMATCH**
The provided version is invalid/unsupported.

enumerator **NVML_ERROR_DEPRECATED**
The requested functionality has been deprecated.

enumerator **NVML_ERROR_NOT_READY**
The system is not ready for the request.

enumerator **NVML_ERROR_GPU_NOT_FOUND**
No GPUs were found.

enumerator **NVML_ERROR_INVALID_STATE**
Resource not in correct state to perform requested operation.

enumerator **NVML_ERROR_RESET_TYPE_NOT_SUPPORTED**
Reset not supported for given device/parameters.

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

enum **nvmlTemperatureSensors_t**
Temperature sensors.

Values:

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

enumerator **NVML_TEMPERATURE_GPU_MAX**
Temperature from the hottest part of the GPU die.

enumerator **NVML_TEMPERATURE_COUNT**

enum **nvmlTemperatureThresholds_t**
Temperature thresholds.

Values:

enumerator **NVML_TEMPERATURE_THRESHOLD_SHUTDOWN**
Temperature at which the GPU will shut down for HW protection.

enumerator **NVML_TEMPERATURE_THRESHOLD_SLOWDOWN**
Temperature at which the GPU will begin HW slowdown.

enumerator **NVML_TEMPERATURE_THRESHOLD_MEM_MAX**
Memory Temperature at which the GPU will begin SW slowdown.

enumerator **NVML_TEMPERATURE_THRESHOLD_GPU_MAX**
GPU Temperature at which the GPU can be throttled below base clock.

enumerator **NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_MIN**
Minimum GPU Temperature that can be set as acoustic threshold.

enumerator **NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_CURR**
Current temperature that is set as acoustic threshold.

enumerator **NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_MAX**
Maximum GPU temperature that can be set as acoustic threshold.

enumerator **NVML_TEMPERATURE_THRESHOLD_GPS_CURR**
Current temperature that is set as gps threshold.

enumerator **NVML_TEMPERATURE_THRESHOLD_COUNT**

5.2.1.6.7 Typedefs

typedef unsigned int **nvmlBusType_t**

typedef *nvmlClockOffset_v1_t* **nvmlClockOffset_t**

typedef unsigned int **nvmlDeviceArchitecture_t**

typedef *nvmlDeviceCurrentClockFreqs_v1_t* **nvmlDeviceCurrentClockFreqs_t**

typedef *nvmlDevicePerfModes_v1_t* **nvmlDevicePerfModes_t**

typedef *nvmlDramEncryptionInfo_v1_t* **nvmlDramEncryptionInfo_t**

typedef *nvmlEccSramErrorStatus_v1_t* **nvmlEccSramErrorStatus_t**

typedef *nvmlEccSramUniqueUncorrectedErrorCounts_v1_t*
nvmlEccSramUniqueUncorrectedErrorCounts_t

typedef unsigned int **nvmlFanControlPolicy_t**

typedef *nvmlFanSpeedInfo_v1_t* **nvmlFanSpeedInfo_t**

typedef *nvmlMarginTemperature_v1_t* **nvmlMarginTemperature_t**

typedef *nvmlPlatformInfo_v2_t* **nvmlPlatformInfo_t**

typedef unsigned char **nvmlPowerScopeType_t**

typedef unsigned int **nvmlPowerSource_t**

typedef *nvmlProcessesUtilizationInfo_v1_t* **nvmlProcessesUtilizationInfo_t**

5.2.1.7 Device Queries

This chapter describes that queries that NVML can perform against each device.

//

In each case the device is identified with an nvmlDevice_t handle. This handle is obtained by calling one of *nvmlDeviceGetHandleByIndex_v2()*, *nvmlDeviceGetHandleBySerial()*, *nvmlDeviceGetHandleByPciBusId_v2()*. or *nvmlDeviceGetHandleByUUID()*.

5.2.1.7.1 CPU and Memory Affinity

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

5.2.1.7.1.1 Macros

NVML_AFFINITY_SCOPE_NODE
Scope of NUMA node for affinity queries.

NVML_AFFINITY_SCOPE_SOCKET
Scope of processor socket for affinity queries.

5.2.1.7.1.2 Functions

nvmlReturn_t *nvmlDeviceClearCpuAffinity*(nvmlDevice_t device)
Clear all affinity bindings for the calling thread.

nvmlReturn_t *nvmlDeviceGetAddressingMode*(nvmlDevice_t device, nvmlDeviceAddressingMode_t *mode)
Get the addressing mode for a given GPU.

nvmlReturn_t *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 This is equivalent to calling *nvmlDeviceGetCpuAffinityWithinScope* with *NVML_AFFINITY_SCOPE_NODE* .

nvmlReturn_t *nvmlDeviceGetCpuAffinityWithinScope*(nvmlDevice_t device, unsigned int cpuSetSize, unsigned long *cpuSet, nvmlAffinityScope_t scope)
Retrieves an array of unsigned ints (sized to cpuSetSize) of bitmasks with the ideal CPU affinity within node or socket for the device.

nvmlReturn_t *nvmlDeviceGetMemoryAffinity*(nvmlDevice_t device, unsigned int nodeSetSize, unsigned long *nodeSet, nvmlAffinityScope_t scope)
Retrieves an array of unsigned ints (sized to nodeSetSize) of bitmasks with the ideal memory affinity within node or socket for the device.

nvmlReturn_t *nvmlDeviceGetNumaNodeId*(nvmlDevice_t device, unsigned int *node)
Get the NUMA node of the given GPU device.

nvmlReturn_t *nvmlDeviceGetRepairStatus*(nvmlDevice_t device, nvmlRepairStatus_t *repairStatus)
Get the repair status for TPC/Channel repair.

nvmlReturn_t *nvmlDeviceGetUnrepairableMemoryFlag_v1*(nvmlDevice_t device, nvmlUnrepairableMemoryStatus_v1_t *unrepairableMemoryStatus)
Get the unrepairable memory flag for a given GPU.

nvmlReturn_t *nvmlDeviceSetCpuAffinity*(nvmlDevice_t device)
Sets the ideal affinity for the calling thread and device using the guidelines given in *nvmlDeviceGetCpuAffinity()* .

5.2.1.7.1.3 Typedefs

nvmlAffinityScope_t

5.2.1.7.1.4 Macros

NVML_AFFINITY_SCOPE_NODE
Scope of NUMA node for affinity queries.

NVML_AFFINITY_SCOPE_SOCKET
Scope of processor socket for affinity queries.

5.2.1.7.1.5 Functions

nvmlReturn_t **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

nvmlReturn_t **nvmlDeviceGetAddressingMode**(
nvmlDevice_t device, *nvmlDeviceAddressingMode_t* *mode,
)

Get the addressing mode for a given GPU.

Addressing modes can be one of:

a. HMM: System allocated memory (malloc, mmap) is addressable from the device (GPU), via software-based mirroring of the CPU's page tables, on the GPU.

b. ATS: System allocated memory (malloc, mmap) is addressable from the device (GPU), via Address Translation Services. This means that there is (effectively) a single set of page tables, and the CPU and GPU both use them.

c. None: Neither HMM nor ATS is active.

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

Parameters

- **device** – **[in]** The device handle
- **mode** – **[out]** Pointer to addressing mode of the device

Returns

- *NVML_SUCCESS* if *mode* is retrieved successfully
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the provided version is invalid/unsupported
- *NVML_ERROR_NOT_SUPPORTED* if request is not supported on the current platform
- *NVML_ERROR_INVALID_ARGUMENT* if *device node* is invalid

nvmlReturn_t **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 This is equivalent to calling *nvmlDeviceGetCpuAffinityWithinScope* with *NVML_AFFINITY_SCOPE_NODE*.

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

Note

In the MPS locality-domain view, this API returns the CPU affinity of the physical parent associated with the locality-domain handle. See *MPS Locality Domain Device Enumeration*.

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

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

Retrieves an array of unsigned ints (sized to cpuSetSize) of bitmasks with the ideal CPU affinity within node or socket for the device.

For example, if processors 0, 1, 32, and 33 are ideal for the device and cpuSetSize == 2, result[0] = 0x3, result[1] = 0x3

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

Note

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

Parameters

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

Returns

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

- *NVML_ERROR_UNKNOWN* on any unexpected error

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

Retrieves an array of unsigned ints (sized to nodeSetSize) of bitmasks with the ideal memory affinity within node or socket for the device.

For example, if NUMA node 0, 1 are ideal within the socket for the device and nodeSetSize == 1, result[0] = 0x3

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

Note

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

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetNumaNodeId**(
nvmlDevice_t device, unsigned int *node,
)

Get the NUMA node of the given GPU device.

This only applies to platforms where the GPUs are NUMA nodes.

Note

In the MPS locality-domain view, this API returns the NUMA node supplied by MPS for the locality domain. See *MPS Locality Domain Device Enumeration*.

Parameters

- **device** – **[in]** The device handle
- **node** – **[out]** NUMA node ID of the device

Returns

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

nvmlReturn_t **nvmlDeviceGetRepairStatus**(
    nvmlDevice_t device, *nvmlRepairStatus_t* *repairStatus,
)

Get the repair status for TPC/Channel repair.

For Ampere or newer fully supported devices.

Parameters

- **device** – **[in]** The identifier of the target device
- **repairStatus** – **[out]** Reference to *nvmlRepairStatus_t*

Returns

- *NVML_SUCCESS* if the query was successful
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the provided version is invalid/unsupported
- *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_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetUnrepairableMemoryFlag_v1**(
    nvmlDevice_t device, *nvmlUnrepairableMemoryStatus_v1_t* *unrepairableMemoryStatus,
)

Get the unrepairable memory flag for a given GPU.

For Hopper or newer fully supported devices.

Parameters

- **device** – **[in]** The identifier of the target device
- **unrepairableMemoryStatus** – **[out]** Reference to nvmlUnrepairableMemoryStatus_v1_t

Returns

- *NVML_SUCCESS* if the query was successful
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the provided version is invalid/unsupported
- *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_UNKNOWN* on any unexpected error

nvmlReturn_t **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 1024 processors.

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 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

5.2.1.7.1.6 Typedefs

typedef unsigned int **nvmlAffinityScope_t**

5.2.1.7.2 Macros

nvmlTemperature_v1

Version macro for *nvmlTemperature_v1_t* .

5.2.1.7.3 Functions

nvmlReturn_t *nvmlDeviceGetAPIRestriction*(nvmlDevice_t device, nvmlRestrictedAPI_t apiType, nvmlEnableState_t *isRestricted)
Retrieves the root/admin permissions on the target API.

nvmlReturn_t *nvmlDeviceGetAdaptiveClockInfoStatus*(nvmlDevice_t device, unsigned int *adaptiveClockStatus)
Gets the device's Adaptive Clock status.

nvmlReturn_t *nvmlDeviceGetAdaptiveTgpModeInfo_v1*(nvmlDevice_t device, nvmlAdaptiveTgpModeInfo_v1_t *info)
Retrieves Adaptive TGP Mode state and telemetry for a GPU.

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

nvmlReturn_t *nvmlDeviceGetArchitecture*(nvmlDevice_t device, nvmlDeviceArchitecture_t *arch)
Get architecture for device.

nvmlReturn_t *nvmlDeviceGetAttributes_v2*(nvmlDevice_t device, nvmlDeviceAttributes_t *attributes)
Get attributes (engine counts etc.) for the given NVML device handle.

nvmlReturn_t *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* .

nvmlReturn_t *nvmlDeviceGetBAR1MemoryInfo*(nvmlDevice_t device, nvmlBAR1Memory_t *bar1Memory)
Gets Total, Available and Used size of BAR1 memory.

nvmlReturn_t *nvmlDeviceGetBBXTimeData_v1*(nvmlDevice_t device, nvmlBBXTimeData_v1_t *timeData)
Retrieves the cumulative number of seconds the GPU has had the driver loaded.

nvmlReturn_t *nvmlDeviceGetBankRemapperStatus_v1*(nvmlDevice_t device, nvmlEccBankRemapperStatus_v1_t *pBankRemapperStatus)
Get bank remapper status.

nvmlReturn_t *nvmlDeviceGetBoardId*(nvmlDevice_t device, unsigned int *boardId)
Retrieves the device boardId from 0-N.

nvmlReturn_t *nvmlDeviceGetBoardPartNumber*(nvmlDevice_t device, char *partNumber, unsigned int length)
Retrieves the the device board part number which is programmed into the board's InfoROM.

nvmlReturn_t *nvmlDeviceGetBrand*(nvmlDevice_t device, nvmlBrandType_t *type)
Retrieves the brand of this device.

nvmlReturn_t *nvmlDeviceGetBridgeChipInfo*(nvmlDevice_t device, nvmlBridgeChipHierarchy_t *bridgeHierarchy)
Get Bridge Chip Information for all the bridge chips on the board.

nvmlReturn_t *nvmlDeviceGetBusType*(nvmlDevice_t device, nvmlBusType_t *type)
Get the type of the GPU Bus (PCIe, PCI, ...)

nvmlReturn_t *nvmlDeviceGetC2cModeInfoV*(nvmlDevice_t device, nvmlC2cModeInfo_v1_t *c2cModeInfo)
Retrieves the Device's C2C Mode information.

nvmlReturn_t *nvmlDeviceGetClkMonStatus*(nvmlDevice_t device, nvmlClkMonStatus_t *status)
Retrieves the frequency monitor fault status for the device.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetClockInfo*(nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)
Retrieves the current clock speeds for the device.

nvmlReturn_t *nvmlDeviceGetClockOffsets*(nvmlDevice_t device, nvmlClockOffset_t *info)
Retrieve min, max and current clock offset of some clock domain for a given PState.

nvmlReturn_t *nvmlDeviceGetComputeMode*(nvmlDevice_t device, nvmlComputeMode_t *mode)
Retrieves the current compute mode for the device or MIG device.

nvmlReturn_t *nvmlDeviceGetComputeRunningProcesses_v3*(nvmlDevice_t device, unsigned int *infoCount, nvmlProcessInfo_t *infos)
Get information about processes with a compute context on a device.

nvmlReturn_t *nvmlDeviceGetConfComputeGpuAttestationReport*(nvmlDevice_t device, nvmlConfComputeGpuAttestationReport_t *gpuAtstReport)
Get Conf Computing GPU attestation report.

nvmlReturn_t *nvmlDeviceGetConfComputeGpuCertificate*(nvmlDevice_t device, nvmlConfComputeGpuCertificate_t *gpuCert)
Get Conf Computing GPU certificate details.

nvmlReturn_t *nvmlDeviceGetConfComputeMemSizeInfo*(nvmlDevice_t device, nvmlConfComputeMemSizeInfo_t *memInfo)
Get Conf Computing Protected and Unprotected Memory Sizes.

nvmlReturn_t *nvmlDeviceGetConfComputeProtectedMemoryUsage*(nvmlDevice_t device, nvmlMemory_t *memory)
Get Conf Computing protected memory usage.

nvmlReturn_t *nvmlDeviceGetCoolerInfo*(nvmlDevice_t device, nvmlCoolerInfo_t *coolerInfo)
Retrieves the cooler's information.

nvmlReturn_t *nvmlDeviceGetCount_v2*(unsigned int *deviceCount)
Retrieves the number of compute devices in the system.

nvmlReturn_t *nvmlDeviceGetCudaComputeCapability*(nvmlDevice_t device, int *major, int *minor)
Retrieves the CUDA compute capability of the device.

nvmlReturn_t *nvmlDeviceGetCurrPcieLinkGeneration*(nvmlDevice_t device, unsigned int *currLinkGen)
Retrieves the current PCIe link generation.

nvmlReturn_t *nvmlDeviceGetCurrPcieLinkWidth*(nvmlDevice_t device, unsigned int *currLinkWidth)
Retrieves the current PCIe link width.

nvmlReturn_t *nvmlDeviceGetCurrentClockFreqs*(nvmlDevice_t device, nvmlDeviceCurrentClockFreqs_t *currentClockFreqs)
Retrieves a string with the associated current GPU Clock and Memory Clock values.

nvmlReturn_t *nvmlDeviceGetCurrentClocksEventReasons*(nvmlDevice_t device, unsigned long long *clocksEventReasons)
Retrieves current clocks event reasons.

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

nvmlReturn_t *nvmlDeviceGetDecoderUtilization*(nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)
Retrieves the current utilization and sampling size in microseconds for the Decoder.

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

nvmlReturn_t *nvmlDeviceGetDefaultEccMode*(nvmlDevice_t device, nvmlEnableState_t *defaultMode)
Retrieves the default ECC modes for the device.

nvmlReturn_t *nvmlDeviceGetDetailedEccErrors*(nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, nvmlEccErrorCounts_t *eccCounts)
Retrieves the detailed ECC error counts for the device.

nvmlReturn_t *nvmlDeviceGetDisplayActive*(nvmlDevice_t device, nvmlEnableState_t *isActive)
Retrieves the display active state for the device.

nvmlReturn_t *nvmlDeviceGetDisplayMode*(nvmlDevice_t device, nvmlEnableState_t *display)
Retrieves the display mode for the device.

nvmlReturn_t *nvmlDeviceGetDramEncryptionMode*(nvmlDevice_t device, nvmlDramEncryptionInfo_t *current, nvmlDramEncryptionInfo_t *pending)
Retrieves the current and pending DRAM Encryption modes for the device.

nvmlReturn_t *nvmlDeviceGetDriverModel_v2*(nvmlDevice_t device, nvmlDriverModel_t *current, nvmlDriverModel_t *pending)
Retrieves the current and pending driver model for the device.

nvmlReturn_t *nvmlDeviceGetDynamicPstatesInfo*(nvmlDevice_t device, nvmlGpuDynamicPstatesInfo_t *pDynamicPstatesInfo)
Retrieve performance monitor samples from the associated subdevice.

nvmlReturn_t *nvmlDeviceGetEccMode*(nvmlDevice_t device, nvmlEnableState_t *current, nvmlEnableState_t *pending)
Retrieves the current and pending ECC modes for the device.

nvmlReturn_t *nvmlDeviceGetEncoderCapacity*(nvmlDevice_t device, nvmlEncoderType_t encoderQueryType, unsigned int *encoderCapacity)
Retrieves the current capacity of the device's encoder, as a percentage of maximum encoder capacity with valid values in the range 0-100.

nvmlReturn_t *nvmlDeviceGetEncoderSessions*(nvmlDevice_t device, unsigned int *sessionCount, nvmlEncoderSessionInfo_t *sessionInfos)
Retrieves information about active encoder sessions on a target device.

nvmlReturn_t *nvmlDeviceGetEncoderStats*(nvmlDevice_t device, unsigned int *sessionCount, unsigned int *averageFps, unsigned int *averageLatency)
Retrieves the current encoder statistics for a given device.

nvmlReturn_t *nvmlDeviceGetEncoderUtilization*(nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)
Retrieves the current utilization and sampling size in microseconds for the Encoder.

nvmlReturn_t *nvmlDeviceGetEnforcedPowerLimit*(nvmlDevice_t device, unsigned int *limit)
Get the effective power limit that the driver enforces after taking into account all limiters.

nvmlReturn_t *nvmlDeviceGetFBCSessions*(nvmlDevice_t device, unsigned int *sessionCount, nvmlFBCSessionInfo_t *sessionInfo)
Retrieves information about active frame buffer capture sessions on a target device.

nvmlReturn_t *nvmlDeviceGetFBCStats*(nvmlDevice_t device, nvmlFBCStats_t *fbcStats)
Retrieves the active frame buffer capture sessions statistics for a given device.

nvmlReturn_t *nvmlDeviceGetFanControlPolicy_v2*(nvmlDevice_t device, unsigned int fan, nvmlFanControlPolicy_t *policy)
Gets current fan control policy.

nvmlReturn_t *nvmlDeviceGetFanSpeed*(nvmlDevice_t device, unsigned int *speed)
Retrieves the intended operating speed of the device's fan.

nvmlReturn_t *nvmlDeviceGetFanSpeedRPM*(nvmlDevice_t device, nvmlFanSpeedInfo_t *fanSpeed)
Retrieves the intended operating speed in rotations per minute (RPM) of the device's specified fan.

nvmlReturn_t *nvmlDeviceGetFanSpeed_v2*(nvmlDevice_t device, unsigned int fan, unsigned int *speed)
Retrieves the intended operating speed of the device's specified fan.

nvmlReturn_t *nvmlDeviceGetGpcClkMinMaxVfOffset*(nvmlDevice_t device, int *minOffset, int *maxOffset)
Retrieve the GPCCLK min max VF offset value.

nvmlReturn_t *nvmlDeviceGetGpcClkVfOffset*(nvmlDevice_t device, int *offset)
Retrieve the GPCCLK VF offset value.

nvmlReturn_t *nvmlDeviceGetGpuFabricInfo*(nvmlDevice_t device, nvmlGpuFabricInfo_t *gpuFabricInfo)

nvmlReturn_t *nvmlDeviceGetGpuFabricInfoV*(nvmlDevice_t device, nvmlGpuFabricInfoV_t *gpuFabricInfo)
Versioned wrapper around *nvmlDeviceGetGpuFabricInfo* that accepts a versioned *nvmlGpuFabricInfo_v2_t* or later output structure.

nvmlReturn_t *nvmlDeviceGetGpuFabricInfo_v4*(nvmlDevice_t device, nvmlGpuFabricInfo_v4_t *gpuFabricInfo)
Retrieves GPU fabric information including per-type clique assignments.

nvmlReturn_t *nvmlDeviceGetGpuMaxPcieLinkGeneration*(nvmlDevice_t device, unsigned int *maxLinkGenDevice)
Retrieves the maximum PCIe link generation supported by this device.

nvmlReturn_t *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).

nvmlReturn_t *nvmlDeviceGetGpuUptime_v1*(nvmlDevice_t device, nvmlGpuUptime_v1_t *uptime)
Retrieves the GPU uptime.

nvmlReturn_t *nvmlDeviceGetGraphicsRunningProcesses_v3*(nvmlDevice_t device, unsigned int *infoCount, nvmlProcessInfo_t *infos)
Get information about processes with a graphics context on a device.

nvmlReturn_t *nvmlDeviceGetGspFirmwareMode*(nvmlDevice_t device, unsigned int *isEnabled, unsigned int *defaultMode)
Retrieve GSP firmware mode.

nvmlReturn_t *nvmlDeviceGetGspFirmwareVersion*(nvmlDevice_t device, char *version)
Retrieve GSP firmware version.

nvmlReturn_t *nvmlDeviceGetHandleByIndex_v2*(unsigned int index, nvmlDevice_t *device)
Acquire the handle for a particular device, based on its index.

nvmlReturn_t *nvmlDeviceGetHandleByPciBusId_v2*(const char *pciBusId, nvmlDevice_t *device)
Acquire the handle for a particular device, based on its PCI bus id.

nvmlReturn_t *nvmlDeviceGetHandleBySerial*(const char *serial, nvmlDevice_t *device)
Acquire the handle for a particular device, based on its board serial number.

nvmlReturn_t *nvmlDeviceGetHandleByUUID*(const char *uuid, nvmlDevice_t *device)
Acquire the handle for a particular device, based on its globally unique immutable UUID (in ASCII format) associated with each device.

nvmlReturn_t *nvmlDeviceGetHandleByUUIDV*(const nvmlUUID_t *uuid, nvmlDevice_t *device)
Acquire the handle for a particular device, based on its globally unique immutable UUID (in either ASCII or binary format) associated with each device.

nvmlReturn_t *nvmlDeviceGetHostname_v1*(nvmlDevice_t device, nvmlHostname_v1_t *hostname)
Get the hostname for the device.

nvmlReturn_t *nvmlDeviceGetIndex*(nvmlDevice_t device, unsigned int *index)
Retrieves the NVML index of this device.

nvmlReturn_t *nvmlDeviceGetInforomConfigurationChecksum*(nvmlDevice_t device, unsigned int *checksum)
Retrieves the checksum of the configuration stored in the device's infoROM.

nvmlReturn_t *nvmlDeviceGetInforomImageVersion*(nvmlDevice_t device, char *version, unsigned int length)
Retrieves the global infoROM image version.

nvmlReturn_t *nvmlDeviceGetInforomVersion*(nvmlDevice_t device, nvmlInforomObject_t object, char *version, unsigned int length)
Retrieves the version information for the device's infoROM object.

nvmlReturn_t *nvmlDeviceGetIrqNum*(nvmlDevice_t device, unsigned int *irqNum)
Gets the device's interrupt number.

nvmlReturn_t *nvmlDeviceGetJpgUtilization*(nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)
Retrieves the current utilization and sampling size in microseconds for the JPG.

nvmlReturn_t *nvmlDeviceGetLastBBXFlushTime*(nvmlDevice_t device, unsigned long long *timestamp, unsigned long *durationUs)
Retrieves the timestamp and the duration of the last flush of the BBX (blackbox) infoROM object during the current run.

nvmlReturn_t *nvmlDeviceGetMPSComputeRunningProcesses_v3*(nvmlDevice_t device, unsigned int *infoCount, nvmlProcessInfo_t *infos)
Get information about processes with a Multi-Process Service (MPS) compute context on a device.

nvmlReturn_t *nvmlDeviceGetMarginTemperature*(nvmlDevice_t device, nvmlMarginTemperature_t *marginTempInfo)
Retrieves the thermal margin temperature (distance to nearest slowdown threshold).

nvmlReturn_t *nvmlDeviceGetMaxClockInfo*(nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)
Retrieves the maximum clock speeds for the device.

nvmlReturn_t *nvmlDeviceGetMaxCustomerBoostClock*(nvmlDevice_t device, nvmlClockType_t clockType, unsigned int *clockMHz)
Retrieves the customer defined maximum boost clock speed specified by the given clock type.

nvmlReturn_t *nvmlDeviceGetMaxPcieLinkGeneration*(nvmlDevice_t device, unsigned int *maxLinkGen)
Retrieves the maximum PCIe link generation possible with this device and system.

nvmlReturn_t *nvmlDeviceGetMaxPcieLinkWidth*(nvmlDevice_t device, unsigned int *maxLinkWidth)
Retrieves the maximum PCIe link width possible with this device and system.

nvmlReturn_t *nvmlDeviceGetMemClkMinMaxVfOffset*(nvmlDevice_t device, int *minOffset, int *maxOffset)
Retrieve the MemClk (Memory Clock) min max VF offset value.

nvmlReturn_t *nvmlDeviceGetMemClkVfOffset*(nvmlDevice_t device, int *offset)
Retrieve the MemClk (Memory Clock) VF offset value.

nvmlReturn_t *nvmlDeviceGetMemoryBusWidth*(nvmlDevice_t device, unsigned int *busWidth)
Gets the device's memory bus width.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetMemoryInfo*(nvmlDevice_t device, nvmlMemory_t *memory)
Retrieves the amount of used, free, reserved and total memory available on the device, in bytes.

nvmlReturn_t *nvmlDeviceGetMemoryInfo_v2*(nvmlDevice_t device, nvmlMemory_v2_t *memory)
Retrieves the amount of used, free, reserved and total memory available on the device, in bytes.

nvmlReturn_t *nvmlDeviceGetMemoryLimits_v1*(nvmlDevice_t device, nvmlGetMemoryLimits_v1_t *limits)
Get the memory limits of the device for the cgroup partition.

nvmlReturn_t *nvmlDeviceGetMinMaxClockOfPState*(nvmlDevice_t device, nvmlClockType_t type, nvmlPstates_t pstate, unsigned int *minClockMHz, unsigned int *maxClockMHz)
Retrieve min and max clocks of some clock domain for a given PState.

nvmlReturn_t *nvmlDeviceGetMinMaxFanSpeed*(nvmlDevice_t device, unsigned int *minSpeed, unsigned int *maxSpeed)
Retrieves the min and max fan speed that user can set for the GPU fan.

nvmlReturn_t *nvmlDeviceGetMinorNumber*(nvmlDevice_t device, unsigned int *minorNumber)
Retrieves minor number for the device.

nvmlReturn_t *nvmlDeviceGetModuleId*(nvmlDevice_t device, unsigned int *moduleId)
Get a unique identifier for the device module on the baseboard.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetName*(nvmlDevice_t device, char *name, unsigned int length)
Retrieves the name of this device.

nvmlReturn_t *nvmlDeviceGetNumFans*(nvmlDevice_t device, unsigned int *numFans)
Retrieves the number of fans on the device.

nvmlReturn_t *nvmlDeviceGetNumGpuCores*(nvmlDevice_t device, unsigned int *numCores)
Gets the device's core count.

nvmlReturn_t *nvmlDeviceGetOfaUtilization*(nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)
Retrieves the current utilization and sampling size in microseconds for the OFA (Optical Flow Accelerator)

nvmlReturn_t *nvmlDeviceGetP2PStatus*(nvmlDevice_t device1, nvmlDevice_t device2, nvmlGpuP2PCapsIndex_t p2pIndex, nvmlGpuP2PStatus_t *p2pStatus)
Retrieve the status for a given p2p capability index between a given pair of GPU.

nvmlReturn_t *nvmlDeviceGetPciInfoExt*(nvmlDevice_t device, nvmlPciInfoExt_t *pci)
Retrieves PCI attributes of this device.

nvmlReturn_t *nvmlDeviceGetPciInfo_v3*(nvmlDevice_t device, nvmlPciInfo_t *pci)
Retrieves the PCI attributes of this device.

nvmlReturn_t *nvmlDeviceGetPcieLinkMaxSpeed*(nvmlDevice_t device, unsigned int *maxSpeed)
Gets the device's PCIE Max Link speed in MBPS.

nvmlReturn_t *nvmlDeviceGetPcieReplayCounter*(nvmlDevice_t device, unsigned int *value)
Retrieve the PCIe replay counter.

nvmlReturn_t *nvmlDeviceGetPcieSpeed*(nvmlDevice_t device, unsigned int *pcieSpeed)
Gets the device's PCIe Link speed in Mbps.

nvmlReturn_t *nvmlDeviceGetPcieThroughput*(nvmlDevice_t device, nvmlPcieUtilCounter_t counter, unsigned int *value)
Retrieve PCIe utilization information.

nvmlReturn_t *nvmlDeviceGetPdi*(nvmlDevice_t device, nvmlPdi_t *pdi)
Retrieves the Per Device Identifier (PDI) associated with this device.

nvmlReturn_t *nvmlDeviceGetPerformanceModes*(nvmlDevice_t device, nvmlDevicePerfModes_t *perfModes)
Retrieves a performance mode string with all the performance modes defined for this device along with their associated GPU Clock and Memory Clock values.

nvmlReturn_t *nvmlDeviceGetPerformanceState*(nvmlDevice_t device, nvmlPstates_t *pState)
Retrieves the current performance state for the device.

nvmlReturn_t *nvmlDeviceGetPersistenceMode*(nvmlDevice_t device, nvmlEnableState_t *mode)
Retrieves the persistence mode associated with this device.

nvmlReturn_t *nvmlDeviceGetPlatformInfo*(nvmlDevice_t device, nvmlPlatformInfo_t *platformInfo)
Get platform information of this device.

nvmlReturn_t *nvmlDeviceGetPowerManagementDefaultLimit*(nvmlDevice_t device, unsigned int *defaultLimit)
Retrieves default power management limit on this device, in milliwatts.

nvmlReturn_t *nvmlDeviceGetPowerManagementLimit*(nvmlDevice_t device, unsigned int *limit)
Retrieves the power management limit associated with this device.

nvmlReturn_t *nvmlDeviceGetPowerManagementLimitConstraints*(nvmlDevice_t device, unsigned int *minLimit, unsigned int *maxLimit)
Retrieves information about possible values of power management limits on this device.

nvmlReturn_t *nvmlDeviceGetPowerManagementMode*(nvmlDevice_t device, nvmlEnableState_t *mode)

nvmlReturn_t *nvmlDeviceGetPowerMizerMode_v1*(nvmlDevice_t device, nvmlDevicePowerMizerModes_v1_t *powerMizerMode)
Retrieves current power mizer mode on this device.

nvmlReturn_t *nvmlDeviceGetPowerSource*(nvmlDevice_t device, nvmlPowerSource_t *powerSource)
Gets the devices power source.

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

nvmlReturn_t *nvmlDeviceGetPowerUsage*(nvmlDevice_t device, unsigned int *power)
Retrieves power usage for this GPU in milliwatts and its associated circuitry (e.g.

nvmlReturn_t *nvmlDeviceGetProcessUtilization*(nvmlDevice_t device, nvmlProcessUtilizationSample_t *utilization, unsigned int *processSamplesCount, unsigned long long lastSeenTimeStamp)
Retrieves the current utilization and process ID.

nvmlReturn_t *nvmlDeviceGetProcessesUtilizationInfo*(nvmlDevice_t device, nvmlProcessesUtilizationInfo_t *procesesUtilInfo)
Retrieves the recent utilization and process ID for all running processes.

nvmlReturn_t *nvmlDeviceGetRemappedRows*(nvmlDevice_t device, unsigned int *corrRows, unsigned int *uncRows, unsigned int *isPending, unsigned int *failureOccurred)
Get number of remapped rows.

nvmlReturn_t *nvmlDeviceGetRemappedRows_v2*(nvmlDevice_t device, nvmlRemappedRowsInfo_v2_t *info)
Get the status of row remapper.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetRetiredPagesPendingStatus*(nvmlDevice_t device, nvmlEnableState_t *isPending)
Check if any pages are pending retirement and need a reboot to fully retire.

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

nvmlReturn_t *nvmlDeviceGetRowRemapperHistogram*(nvmlDevice_t device, nvmlRowRemapperHistogramValues_t *values)
Get the row remapper histogram.

nvmlReturn_t *nvmlDeviceGetRunningProcessDetailList*(nvmlDevice_t device, nvmlProcessDetailList_t *plist)
Get information about running processes on a device for input context.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetSerial*(nvmlDevice_t device, char *serial, unsigned int length)
Retrieves the globally unique board serial number associated with this device's board.

nvmlReturn_t *nvmlDeviceGetSramEccErrorStatus*(nvmlDevice_t device, nvmlEccSramErrorStatus_t *status)
Get SRAM ECC error status of this device.

nvmlReturn_t *nvmlDeviceGetSramUniqueUncorrectedEccErrorCounts*(nvmlDevice_t device, nvmlEccSramUniqueUncorrectedErrorCounts_t *errorCounts)
Retrieves the counts of SRAM unique uncorrected ECC errors.

nvmlReturn_t *nvmlDeviceGetSupportedClocksEventReasons*(nvmlDevice_t device, unsigned long long *supportedClocksEventReasons)
Retrieves bitmask of supported clocks event reasons that can be returned by *nvmlDeviceGetCurrentClocksEventReasons* .

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

nvmlReturn_t *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 *nvmlDeviceSetGpuLockedClocks* .

nvmlReturn_t *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 *nvmlDeviceSetMemoryLockedClocks* .

nvmlReturn_t *nvmlDeviceGetSupportedPerformanceStates*(nvmlDevice_t device, nvmlPstates_t *pstates, unsigned int size)
Get all supported Performance States (P-States) for the device.

nvmlReturn_t *nvmlDeviceGetTargetFanSpeed*(nvmlDevice_t device, unsigned int fan, unsigned int *targetSpeed)
Retrieves the intended target speed of the device's specified fan.

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

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetTemperatureV*(nvmlDevice_t device, nvmlTemperature_t *temperature)
Retrieves the current temperature readings (in degrees C) for the given device.

nvmlReturn_t *nvmlDeviceGetThermalSettings*(nvmlDevice_t device, unsigned int sensorIndex, nvmlGpuThermalSettings_t *pThermalSettings)
Used to execute a list of thermal system instructions.

nvmlReturn_t *nvmlDeviceGetTopologyCommonAncestor*(nvmlDevice_t device1, nvmlDevice_t device2, nvmlGpuTopologyLevel_t *pathInfo)
Retrieve the common ancestor for two devices For all products.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetTotalEccErrors*(nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, unsigned long long *eccCounts)
Retrieves the total ECC error counts for the device.

nvmlReturn_t *nvmlDeviceGetTotalEnergyConsumption*(nvmlDevice_t device, unsigned long long *energy)
Retrieves total energy consumption for this GPU in millijoules (mJ) since the driver was last reloaded.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetUtilizationRates*(nvmlDevice_t device, nvmlUtilization_t *utilization)
Retrieves the current utilization rates for the device's major subsystems.

nvmlReturn_t *nvmlDeviceGetVbiosVersion*(nvmlDevice_t device, char *version, unsigned int length)
Get VBIOS version of the device.

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

nvmlReturn_t *nvmlDeviceOnSameBoard*(nvmlDevice_t device1, nvmlDevice_t device2, int *onSameBoard)
Check if the GPU devices are on the same physical board.

nvmlReturn_t *nvmlDevicePerfMetricsGetSamples_v1*(nvmlDevice_t device, nvmlPerfMetricsSamples_v1_t *samples)
Get Performance Metric samples.

nvmlReturn_t *nvmlDeviceSetAdaptiveTgpMode_v1*(nvmlDevice_t device, nvmlEnableState_t mode)
Request to enable or disable Adaptive TGP Mode for a GPU.

nvmlReturn_t *nvmlDeviceSetClockOffsets*(nvmlDevice_t device, nvmlClockOffset_t *info)
Control current clock offset of some clock domain for a given PState.

nvmlReturn_t *nvmlDeviceSetConfComputeUnprotectedMemSize*(nvmlDevice_t device, unsigned long long sizeKiB)
Set Conf Computing Unprotected Memory Size.

nvmlReturn_t *nvmlDeviceSetDramEncryptionMode*(nvmlDevice_t device, const nvmlDramEncryptionInfo_t *dramEncryption)
Set the DRAM Encryption mode for the device.

nvmlReturn_t *nvmlDeviceSetHostname_v1*(nvmlDevice_t device, nvmlHostname_v1_t *hostname)
Set the hostname for the device.

nvmlReturn_t *nvmlDeviceSetMemoryLimits_v1*(nvmlDevice_t device, nvmlSetMemoryLimits_v1_t *limits)
Set the memory limits of the device for the cgroup partition.

nvmlReturn_t *nvmlDeviceSetPowerManagementLimit_v2*(nvmlDevice_t device, nvmlPowerValue_v2_t *powerValue)
Set new power limit of this device.

nvmlReturn_t *nvmlDeviceSetPowerMizerMode_v1*(nvmlDevice_t device, nvmlDevicePowerMizerModes_v1_t *powerMizerMode)
Sets the new power mizer mode.

nvmlReturn_t *nvmlDeviceValidateInforom*(nvmlDevice_t device)
Reads the infoROM from the flash and verifies the checksums.

nvmlReturn_t *nvmlSystemGetConfComputeCapabilities*(nvmlConfComputeSystemCaps_t *capabilities)
Get Conf Computing System capabilities.

nvmlReturn_t *nvmlSystemGetConfComputeGpusReadyState*(unsigned int *isAcceptingWork)
Get Conf Computing GPUs ready state.

nvmlReturn_t *nvmlSystemGetConfComputeKeyRotationThreshold-Info*(nvmlConfComputeGetKeyRotationThresholdInfo_t *pKeyRotationThrInfo)
Get Conf Computing key rotation threshold detail.

nvmlReturn_t *nvmlSystemGetConfComputeSettings*(nvmlSystemConfComputeSettings_t *settings)
Get Conf Computing System Settings.

nvmlReturn_t *nvmlSystemGetConfComputeSettings_v2*(nvmlSystemConfComputeSettings_v2_t *settings)
Get Conf Computing System Settings, version 2.

nvmlReturn_t *nvmlSystemGetConfComputeState*(nvmlConfComputeSystemState_t *state)
Get Conf Computing System State.

nvmlReturn_t *nvmlSystemSetConfComputeGpusReadyState*(unsigned int isAcceptingWork)
Set Conf Computing GPUs ready state.

nvmlReturn_t *nvmlSystemSetConfComputeKeyRotationThresholdInfo*(nvmlConfComputeSetKeyRotationThresholdInfo_t *pKeyRotationThrInfo)
Set Conf Computing key rotation threshold.

5.2.1.7.4 Groups

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

5.2.1.7.5 Structs

nvmlTemperature_v1_t
Structure used to encapsulate temperature info.

5.2.1.7.6 Typedefs

nvmlTemperature_t

5.2.1.7.7 Macros

nvmlTemperature_v1
Version macro for nvmlTemperature_v1_t.

5.2.1.7.8 Functions

nvmlReturn_t **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.

See also

nvmlRestrictedAPI_t

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

nvmlReturn_t **nvmlDeviceGetAdaptiveClockInfoStatus**(
nvmlDevice_t device, unsigned int *adaptiveClockStatus,
)

Gets the device's Adaptive Clock status.

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetAdaptiveTgpModeInfo_v1**(
nvmlDevice_t device, *nvmlAdaptiveTgpModeInfo_v1_t* *info,
)

Retrieves Adaptive TGP Mode state and telemetry for a GPU.

RUBIN_OR_NEWER%

Populates *info* with the in-band request, out-of-band enablement status, out-of-band override status, arbitrated enablement state, and adjusted TGP power limit. The adjusted TGP power is valid only when feature is enabled. See *nvmlAdaptiveTgpModeInfo_v1_t* for field details.

See also

nvmlDeviceSetAdaptiveTgpMode_v1()

Parameters

- **device** – The identifier of the target device
- **info** – Reference in which to return the Adaptive TGP Mode 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 *device* is invalid or *info* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support Adaptive TGP Mode
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

Deprecated:

Applications clocks are deprecated and will be removed in CUDA 14.0.

nvmlReturn_t **nvmlDeviceGetArchitecture**(
    nvmlDevice_t device, *nvmlDeviceArchitecture_t* *arch,
)

Get architecture for device.

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetAttributes_v2**(
    nvmlDevice_t device, *nvmlDeviceAttributes_t* *attributes,
)

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

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

Note

This API currently only supports MIG device handles.

Parameters

- **device** – NVML device handle
- **attributes** – Device attributes

Returns

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

nvmlReturn_t **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 or newer fully supported devices.

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

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

Parameters

- **device** – The identifier of the target device
- **isEnabled** – Where to store the current state of Auto Boosted clocks of the target device
- **defaultIsEnabled** – Where to store the default Auto Boosted clocks behavior of the target device that the device will revert to when no applications are using the GPU

Returns

- *NVML_SUCCESS* If *isEnabled* has been been set with the Auto Boosted clocks state of *device*

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

nvmlReturn_t **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 or newer fully supported devices.

Note

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

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

nvmlReturn_t **nvmlDeviceGetBBXTimeData_v1**(
    nvmlDevice_t device, *nvmlBBXTimeData_v1_t* *timeData,
)

Retrieves the cumulative number of seconds the GPU has had the driver loaded.

For all products with an inforom.

Parameters

- **device** – The identifier of the target device
- **timeData** – Reference in which to return the cumulative number of seconds the GPU has had the driver loaded

Returns

- *NVML_SUCCESS* if *timeData* has been set
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *timeData* 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

nvmlReturn_t **nvmlDeviceGetBankRemapperStatus_v1**(
nvmlDevice_t device, *nvmlEccBankRemapperStatus_v1_t* *pBankRemapperStatus,
)

Get bank remapper status.

RUBIN_OR_NEWER%

Parameters

- **device** – The identifier of the target device
- **pBankRemapperStatus** – Reference to *nvmlEccBankRemapperStatus_v1_t*

Returns

- *NVML_SUCCESS* if *pBankRemapperStatus* was populated
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *pBankRemapperStatus* 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

nvmlReturn_t **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 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

nvmlReturn_t **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

nvmlReturn_t **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

nvmlReturn_t **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

nvmlReturn_t **nvmlDeviceGetBusType**(
nvmlDevice_t device, *nvmlBusType_t* *type,
)

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

return

- *NVML_SUCCESS* if the bus *type* is successfully retrieved
- *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_UNKNOWN* on any unexpected error

Parameters

- **device** – The identifier of the target device
- **type** – The PCI Bus type

nvmlReturn_t **nvmlDeviceGetC2cModeInfoV**(
nvmlDevice_t device, *nvmlC2cModeInfo_v1_t* *c2cModeInfo,
)

Retrieves the Device's C2C Mode information.

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetClkMonStatus**(
nvmlDevice_t device, *nvmlClkMonStatus_t* *status,
)

Retrieves the frequency monitor fault status for the device.

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

See *nvmlClkMonStatus_t* for details on decoding the status output.

See also

nvmlDeviceGetClkMonStatus()

Parameters

- **device** – The identifier of the target device
- **status** – Reference in which to return the clkmon fault status

Returns

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

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

nvmlReturn_t **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 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

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

Retrieves the current clock speeds for the device.

For Fermi 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

nvmlReturn_t **nvmlDeviceGetClockOffsets**(
    nvmlDevice_t device, *nvmlClockOffset_t* *info,
)

Retrieve min, max and current clock offset of some clock domain for a given PState.

For Maxwell or newer fully supported devices.

Note: *nvmlDeviceGetGpcClkVfOffset*, *nvmlDeviceGetMemClkVfOffset*, *nvmlDeviceGetGpcClkMinMaxVfOffset* and *nvmlDeviceGetMemClkMinMaxVfOffset* will be deprecated in a future release. Use *nvmlDeviceGetClockOffsets* instead.

Parameters

- **device** – The identifier of the target device
- **info** – Structure specifying the clock type (input) and the pstate (input) retrieved clock offset value (output), min clock offset (output) and max clock offset (output)

Returns

- *NVML_SUCCESS* If everything worked
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *type* or *pstate* are invalid or both *minClockOffsetMHz* and *maxClockOffsetMHz* are NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the provided version is invalid/unsupported
- *NVML_ERROR_NOT_SUPPORTED* If the device does not support this feature

nvmlReturn_t **nvmlDeviceGetComputeMode**(
    nvmlDevice_t device, *nvmlComputeMode_t* *mode,
)

Retrieves the current compute mode for the device or MIG device.

For all products.

See *nvmlComputeMode_t* for details on allowed compute modes.

See also

nvmlDeviceSetComputeMode()

Note

If MIG is enabled on a GPU, device must be MIG device handle.

Parameters

- **device** – The identifier of the target device handle or MIG device handle
- **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

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

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

For Fermi or newer fully supported devices.

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

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

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

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

See also

nvmlSystemGetProcessName

Note

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

MIG device handles. Querying per-instance information using MIG device handles is not supported if the device is in vGPU Host virtualization mode.

Parameters

- **device** – The device handle or MIG device handle
- **infoCount** – Reference in which to provide the *infos* array size, and to return the number of returned elements
- **infos** – Reference in which to return the process information

Returns

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

nvmlReturn_t **nvmlDeviceGetConfComputeGpuAttestationReport**(
nvmlDevice_t device, *nvmlConfComputeGpuAttestationReport_t* *gpuAtstReport,
)

Get Conf Computing GPU attestation report.

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

Parameters

- **device** – The identifier of the target device
- **gpuAtstReport** – Reference in which to return the gpu attestation report

Returns

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

nvmlReturn_t **nvmlDeviceGetConfComputeGpuCertificate**(
    nvmlDevice_t device, *nvmlConfComputeGpuCertificate_t* *gpuCert,
)

Get Conf Computing GPU certificate details.

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

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetConfComputeMemSizeInfo**(
    nvmlDevice_t device, *nvmlConfComputeMemSizeInfo_t* *memInfo,
)

Get Conf Computing Protected and Unprotected Memory Sizes.

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

Parameters

- **device** – Device handle
- **memInfo** – Protected/Unprotected Memory sizes

Returns

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

nvmlReturn_t **nvmlDeviceGetConfComputeProtectedMemoryUsage**(
    nvmlDevice_t device, *nvmlMemory_t* *memory,
)

Get Conf Computing protected memory usage.

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

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetCoolerInfo**(
nvmlDevice_t device, *nvmlCoolerInfo_t* *coolerInfo,
)

Retrieves the cooler's information.

Returns a cooler's control signal characteristics. The possible types are restricted, Variable and Toggle. See *nvmlCoolerControl_t* for details on available signal types. Returns objects that cooler cools. Targets may be GPU, Memory, Power Supply or All of these. See *nvmlCoolerTarget_t* for details on available targets.

For Maxwell or newer fully supported devices.

For all discrete products with dedicated fans.

Parameters

- **device** – **[in]** The identifier of the target device
- **coolerInfo** – **[out]** Structure specifying the cooler's control signal characteristics (out) and the target that cooler cools (out)

Returns

- *NVML_SUCCESS* If everything worked
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid, *signalType* or *target* is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the provided version is invalid/unsupported
- *NVML_ERROR_NOT_SUPPORTED* If the *device* does not support this feature

nvmlReturn_t **nvmlDeviceGetCount_v2**(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.

Note

In the MPS locality-domain view, this API returns the number of MPS-reported device entries. Each locality domain has its own NVML index and `nvmlDevice_t` handle. See *MPS Locality Domain Device Enumeration*.

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

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

Retrieves the CUDA compute capability of the device.

For all products.

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

Note

In the MPS locality-domain view, this API returns the compute capability supplied by MPS for the locality domain. See *MPS Locality Domain Device Enumeration*.

Parameters

- **`device`** – The identifier of the target device
- **`major`** – Reference in which to return the major CUDA compute capability
- **`minor`** – Reference in which to return the minor CUDA compute capability

Returns

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

nvmlReturn_t **nvmlDeviceGetCurrPcieLinkGeneration**(
    nvmlDevice_t device, unsigned int *currLinkGen,
)

Retrieves the current PCIe link generation.

For Fermi 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

nvmlReturn_t **nvmlDeviceGetCurrPcieLinkWidth**(
    nvmlDevice_t device, unsigned int *currLinkWidth,
)

Retrieves the current PCIe link width.

For Fermi 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

nvmlReturn_t **nvmlDeviceGetCurrentClockFreqs**(
    nvmlDevice_t device, *nvmlDeviceCurrentClockFreqs_t* *currentClockFreqs,
)

Retrieves a string with the associated current GPU Clock and Memory Clock values.

Not all tokens will be reported on all GPUs, and additional tokens may be added in the future.

Note: These clock values take into account the offset set by clients through /ref nvmlDeviceSetClockOffsets.

Clock values are returned as a comma-separated list of “token=value” pairs. Valid tokens:

Token Value “perf” unsigned int - the Performance level “nvclock” unsigned int - the GPU clocks (in MHz) for the perf level “nvclockmin” unsigned int - the GPU clocks min (in MHz) for the perf level “nvclockmax” unsigned int - the GPU clocks max (in MHz) for the perf level “nvclockeditable” unsigned int - if the GPU clock domain is editable for the perf level “memclock” unsigned int - the memory clocks (in MHz) for the perf level “memclockmin” unsigned int - the memory clocks min (in MHz) for the perf level “memclockmax” unsigned int - the memory clocks max (in MHz) for the perf level “memclockeditable” unsigned int - if the memory clock domain is editable for the perf level “memtransferrate” unsigned int - the memory transfer rate (in MHz) for the perf level “memtransferratemin” unsigned int - the memory transfer rate min (in MHz) for the perf level “memtransferratemax” unsigned int - the memory transfer rate max (in MHz) for the perf level “memtransferrateeditable” unsigned int - if the memory transfer rate is editable for the perf level

Example:

nvclock=324, nvclockmin=324, nvclockmax=324, nvclockeditable=0, memclock=324, memclockmin=324, memclockmax=324, memclockeditable=0, memtransferrate=648, memtransferratemin=648, memtransferratemax=648, memtransferrateeditable=0 ;

Parameters

- **`device`** – The identifier of the target device
- **`currentClockFreqs`** – Reference in which to return the performance level string

Returns

- *NVML_SUCCESS* if *currentClockFreqs* 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

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

Retrieves current clocks event reasons.

For all fully supported products.

See also

NvmlClocksEventReasons

See also

nvmlDeviceGetSupportedClocksEventReasons

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
- **clocksEventReasons** – Reference in which to return bitmask of active clocks event reasons

Returns

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

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

Deprecated:

Use *nvmlDeviceGetCurrentClocksEventReasons* instead

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

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

For Kepler or newer fully supported devices.

Note

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

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

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

Deprecated:

Applications clocks are deprecated and will be removed in CUDA 14.0.

nvmlReturn_t **nvmlDeviceGetDefaultEccMode**(
nvmlDevice_t device, *nvmlEnableState_t* *defaultMode,
)

Retrieves the default ECC modes for the device.

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

See *nvmlEnableState_t* for details on allowed modes.

See also

nvmlDeviceSetEccMode()

Parameters

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

Returns

- *NVML_SUCCESS* if *current* and *pending* have been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *default* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature

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

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

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

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

See *nvmlMemoryErrorType_t* for a description of available bit types.

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

See *nvmlEccErrorCounts_t* for a description of provided detailed ECC counts.

See also

nvmlDeviceClearEccErrorCounts()

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

nvmlReturn_t **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

nvmlReturn_t **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

nvmlReturn_t **nvmlDeviceGetDramEncryptionMode**(
nvmlDevice_t device, *nvmlDramEncryptionInfo_t* *current, *nvmlDramEncryptionInfo_t* *pending,
)

Retrieves the current and pending DRAM Encryption modes for the device.

For Blackwell or newer fully supported devices. Only applicable to devices that support DRAM Encryption Requires *NVML_INFOROM_DEN* version 1.0 or higher.

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

See *nvmlEnableState_t* for details on allowed modes.

See also

nvmlDeviceSetDramEncryptionMode()

Parameters

- **device** – The identifier of the target device
- **current** – Reference in which to return the current DRAM Encryption mode
- **pending** – Reference in which to return the pending DRAM Encryption 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_ARGUMENT_VERSION_MISMATCH* if the argument version is not supported
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetDriverModel_v2**(
nvmlDevice_t device, *nvmlDriverModel_t* *current, *nvmlDriverModel_t* *pending,
)

Retrieves the current and pending driver model for the device.

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

On Windows platforms the device driver can run in either WDDM, MCDM or WDM (TCC) modes. If a display is attached to the device it must run in WDDM mode. MCDM mode is preferred if a display is not attached. TCC mode is deprecated. Driver-model availability is architecture-specific; attempting to set an unsupported driver model returns NVML_ERROR_NOT_SUPPORTED.

See *nvmlDriverModel_t* for details on available driver models.

See also

nvmlDeviceSetDriverModel_v2()

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

nvmlReturn_t **nvmlDeviceGetDynamicPstatesInfo**(
    nvmlDevice_t device, *nvmlGpuDynamicPstatesInfo_t* *pDynamicPstatesInfo,
)

Retrieve performance monitor samples from the associated subdevice.

Parameters

- **device** –
- **pDynamicPstatesInfo** –

Returns

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

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

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

Retrieves the current and pending ECC modes for the device.

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

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

See *nvmlEnableState_t* for details on allowed modes.

See also

nvmlDeviceSetEccMode()

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

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

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

For Maxwell or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **encoderQueryType** – Type of encoder to query
- **encoderCapacity** – Reference to an unsigned int for the encoder capacity

Returns

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

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

Retrieves information about active encoder sessions on a target device.

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

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

For Maxwell or newer fully supported devices.

Parameters

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

Returns

- *NVML_SUCCESS* if *sessionInfos* is fetched
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INSUFFICIENT_SIZE* if *sessionCount* is too small, array element count is returned in *sessionCount*
- *NVML_ERROR_INVALID_ARGUMENT* if *sessionCount* is NULL.

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

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

Retrieves the current encoder statistics for a given device.

For Maxwell or newer fully supported devices.

Parameters

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

Returns

- *NVML_SUCCESS* if *sessionCount*, *averageFps* and *averageLatency* is fetched
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *sessionCount*, or *device* or *averageFps*, or *averageLatency* is NULL
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

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

For Kepler or newer fully supported devices.

Note

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

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

nvmlReturn_t **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 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

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

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

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

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

*sessionCount = 0. The code will return NVML_SUCCESS with number of active FBC sessions updated in *sessionCount.

For Maxwell or newer fully supported devices.

Note

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

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetFBCStats**(
    nvmlDevice_t device, *nvmlFBCStats_t* *fbcStats,
)

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

For Maxwell or newer fully supported devices.

Parameters

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

Returns

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

- *NVML_ERROR_UNKNOWN* on any unexpected error

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

Gets current fan control policy.

For Maxwell or newer fully supported devices.

For all cuda-capable discrete products with fans

Parameters

- **device** – The identifier of the target *device*
- **fan** – The index of the target fan, zero indexed.
- **policy** – Reference in which to return the fan control *policy*

Returns

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

nvmlReturn_t **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 percentage of the product's maximum noise tolerance fan speed. This value may exceed 100% in certain cases.

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

nvmlReturn_t **nvmlDeviceGetFanSpeedRPM**(
nvmlDevice_t device, *nvmlFanSpeedInfo_t* *fanSpeed,
)

Retrieves the intended operating speed in rotations per minute (RPM) of the device's specified fan.

For Maxwell or newer fully supported devices.

For all discrete products with dedicated fans.

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.

Parameters

- **device** – The identifier of the target device
- **fanSpeed** – Structure specifying the index of the target fan (input) and retrieved fan speed value (output)

Returns

- *NVML_SUCCESS* If everything worked
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid, *fan* is not an acceptable index, or *speed* is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the provided version is invalid/unsupported
- *NVML_ERROR_NOT_SUPPORTED* If the *device* does not support this feature

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

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

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

For all discrete products with dedicated fans.

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

Parameters

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

Returns

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

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

Retrieve the GPCCLK min max VF offset value.

Parameters

- **device** – **[in]** The identifier of the target device
- **minOffset** – **[out]** The retrieved GPCCLK VF min offset value
- **maxOffset** – **[out]** The retrieved GPCCLK VF max offset value

Returns

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

nvmlReturn_t **nvmlDeviceGetGpcClkVfOffset**(
    nvmlDevice_t device, int *offset,
)

Retrieve the GPCCLK VF offset value.

Parameters

- **device** – **[in]** The identifier of the target device
- **offset** – **[out]** The retrieved GPCCLK VF offset value

Returns

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

nvmlReturn_t **nvmlDeviceGetGpuFabricInfo**(
    nvmlDevice_t device, *nvmlGpuFabricInfo_t* *gpuFabricInfo,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlDeviceGetGpuFabricInfoV* instead

Get fabric information associated with the device.

For Hopper or newer fully supported devices.

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

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetGpuFabricInfoV**(
    nvmlDevice_t device, *nvmlGpuFabricInfoV_t* *gpuFabricInfo,
)

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

For Hopper or newer fully supported devices.

Note

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

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

Parameters

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

Returns

- *NVML_SUCCESS* Upon success

- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't support gpu fabric

nvmlReturn_t **nvmlDeviceGetGpuFabricInfo_v4**(
    nvmlDevice_t device, *nvmlGpuFabricInfo_v4_t* *gpuFabricInfo,
)

Retrieves GPU fabric information including per-type clique assignments.

Returns fabric clique data via *nvmlGpuFabricInfo_v4_t*. Each entry in the *cliques* array is a (type, id) pair representing a single clique assignment. The number of valid entries is given by *numCliques*. Entries are sorted by ascending type (NVML_GPU_FABRIC_CLIQUE_TYPE_*), then by ascending clique id within each type.

On Hopper systems, the driver reports Unicast Pointer and Multicast Pointer cliques. On Blackwell and Rubin, Unicast Logical Endpoint and Multicast Logical Endpoint are additionally reported.

```
nvmlGpuFabricInfo_v4_t fabricInfo = {0};
nvmlReturn_t result = nvmlDeviceGetGpuFabricInfo_v4(device, &fabricInfo);
```

For Hopper or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **gpuFabricInfo** – Information about GPU fabric state including per-type cliques

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't support gpu fabric
- *NVML_ERROR_INVALID_ARGUMENT* If *device* or *gpuFabricInfo* is invalid

nvmlReturn_t **nvmlDeviceGetGpuMaxPcieLinkGeneration**(
    nvmlDevice_t device, unsigned int *maxLinkGenDevice,
)

Retrieves the maximum PCIe link generation supported by this device.

For Fermi or newer fully supported devices.

Parameters

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

Returns

- *NVML_SUCCESS* if *maxLinkGenDevice* has been populated
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *maxLinkGenDevice* is null
- *NVML_ERROR_NOT_SUPPORTED* if PCIe link information is not available

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

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

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

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

See also

nvmlGpuOperationMode_t

See also

nvmlDeviceSetGpuOperationMode

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

nvmlReturn_t **nvmlDeviceGetGpuUptime_v1**(
nvmlDevice_t device, *nvmlGpuUptime_v1_t* *uptime,
)

Retrieves the GPU uptime.

The uptime denotes the number of seconds elapsed since the GPU was initialized by the driver. The uptime resets to zero on a host reboot, driver unload/reload, or a GPU reset.

For MIG device handles, the parent GPU uptime is returned.

Parameters

- **device** – The identifier of the target device
- **uptime** – Reference in which to return the GPU uptime

Returns

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

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

Get information about processes with a graphics context on a device.

For Kepler or newer fully supported devices.

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

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

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

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

See also

nvmlSystemGetProcessName

Note

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

Parameters

- **device** – The device handle or MIG device handle

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

Returns

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

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

Retrieve GSP firmware mode.

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

Parameters

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

Returns

- *NVML_SUCCESS* if GSP firmware mode is successfully retrieved
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or any of *isEnabled* or *defaultMode* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if GSP firmware is not enabled for GPU
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetGspFirmwareVersion**(
    nvmlDevice_t device, char *version,
)

Retrieve GSP firmware version.

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

Parameters

- **device** – Device handle
- **version** – The retrieved GSP firmware version

Returns

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

nvmlReturn_t **nvmlDeviceGetHandleByIndex_v2**(
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_v2()*. For example, if *accessibleDevices* is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

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

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

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

- The target GPU is an SLI slave

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

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

See also

nvmlDeviceGetIndex

See also

nvmlDeviceGetCount

Note

In the MPS locality-domain view, this API returns the locality-domain handle associated with *index*. See *MPS Locality Domain Device Enumeration*.

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetHandleByPciBusId_v2**(
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_v3()*.

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

- The target GPU is an SLI slave

Note

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

Note

In the MPS locality-domain view, multiple locality domains can share a PCI bus ID. This API returns the first matching handle in NVML enumeration order. Use UUID or index to select a specific locality domain. See *MPS Locality Domain Device Enumeration*.

Parameters

- **pciBusId** – The PCI bus id of the target GPU Accept the following formats (all numbers in hexadecimal): domain:bus:device.function in format x:x:x.x domain:bus:device in format x:x:x bus:device.function in format x:x.x
- **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

nvmlReturn_t **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

See also

nvmlDeviceGetSerial

See also

nvmlDeviceGetHandleByUUID

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

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

Acquire the handle for a particular device, based on its globally unique immutable UUID (in ASCII format) associated with each device.

For all products.

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

See also

nvmlDeviceGetUUID

Note

In the MPS locality-domain view, this API returns the locality-domain handle whose MPS-provided UUID matches *uuid*. See *MPS Locality Domain Device Enumeration*.

Parameters

- **uuid** – The UUID of the target GPU or MIG instance
- **device** – Reference in which to return the device handle or MIG device handle

Returns

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

nvmlReturn_t **nvmlDeviceGetHandleByUUIDV**(
    const *nvmlUUID_t* *uuid, *nvmlDevice_t* *device,
)

Acquire the handle for a particular device, based on its globally unique immutable UUID (in either ASCII or binary format) associated with each device.

See *nvmlUUID_v1_t* for more information on the UUID struct. The caller must set the appropriate version prior to calling this API.

For all products.

This API causes NVML to initialize the target GPU NVML may initialize additional GPUs as it searches for the target GPU

Note

This API returns *NVML_ERROR_NOT_SUPPORTED* in the MPS locality-domain view. Use *nvmlDeviceGetHandleByUUID()* with the ASCII UUID instead. See *MPS Locality Domain Device Enumeration*.

Parameters

- **uuid** – **[in]** The UUID of the target GPU or MIG instance

- **device** – **[out]** Reference in which to return the device handle or MIG device handle

Returns

- *NVML_SUCCESS* if *device* has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *uuid* is invalid, *device* is null or *uuid->type* is invalid
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the provided version is invalid/unsupported
- *NVML_ERROR_NOT_FOUND* if *uuid* does not match a valid device on the system
- *NVML_ERROR_GPU_IS_LOST* if any GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetHostname_v1**(
    nvmlDevice_t device, *nvmlHostname_v1_t* *hostname,
)

Get the hostname for the device.

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

Retrieves the hostname string for the GPU device that was set using *nvmlDeviceSetHostname_v1()*.

See also

nvmlDeviceSetHostname_v1()

Parameters

- **device** – The identifier of the target device
- **hostname** – Reference to the caller-provided *nvmlHostname_v1_t* struct to return the hostname

Returns

- *NVML_SUCCESS* if the hostname was retrieved successfully
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *hostname* 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

nvmlReturn_t **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_v2()*. For example, if *accessibleDevices* is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

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

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

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

See also

nvmlDeviceGetHandleByIndex()

See also

nvmlDeviceGetCount()

Note

In the MPS locality-domain view, this API returns the NVML index of the locality-domain handle. See *MPS Locality Domain Device Enumeration*.

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

nvmlReturn_t **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

nvmlReturn_t **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.

See also

nvmlDeviceGetInforomVersion

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

nvmlReturn_t **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.

See also

nvmlDeviceGetInforomImageVersion

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

nvmlReturn_t **nvmlDeviceGetIrqNum**(
    nvmlDevice_t device, unsigned int *irqNum,
)

Gets the device's interrupt number.

Parameters

- **device** – The identifier of the target device
- **irqNum** – The interrupt number associated with the specified device

Returns

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

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

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

For Turing or newer fully supported devices.

Note

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

Parameters

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

Returns

- *NVML_SUCCESS* if *utilization* has been populated
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, *utilization* is NULL, or *samplingPeriodUs* is NULL

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

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

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

For all products with an inforom.

See also

nvmlDeviceGetInforomVersion

Parameters

- **device** – The identifier of the target device
- **timestamp** – The start timestamp of the last BBX Flush
- **durationUs** – The duration (us) of the last BBX Flush

Returns

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

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

Get information about processes with a Multi-Process Service (MPS) compute context on a device.

For Volta or newer fully supported devices.

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

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

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

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

See also

nvmlSystemGetProcessName

Note

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

Parameters

- **`device`** – The device handle or MIG device handle
- **`infoCount`** – Reference in which to provide the *infos* array size, and to return the number of returned elements
- **`infos`** – Reference in which to return the process information

Returns

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

nvmlReturn_t **`nvmlDeviceGetMarginTemperature`**(
    nvmlDevice_t device, *nvmlMarginTemperature_t* *marginTempInfo,
)

Retrieves the thermal margin temperature (distance to nearest slowdown threshold).

Parameters

- **`device`** – **[in]** The identifier of the target device
- **`marginTempInfo`** – **[inout]** Versioned structure in which to return the temperature reading

Returns

- *NVML_SUCCESS* if the margin temperature was retrieved successfully
- *NVML_ERROR_NOT_SUPPORTED* if request is not supported on the current platform
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *temperature* is NULL
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the right versioned structure is not used
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

Retrieves the maximum clock speeds for the device.

For Fermi or newer fully supported devices.

See *nvmlClockType_t* for details on available clock information.

Note

Current P0 clocks (reported by *nvmlDeviceGetClockInfo*) can differ from max clocks by a 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

nvmlReturn_t **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 Pascal 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 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

nvmlReturn_t **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 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

nvmlReturn_t **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 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

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

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

Parameters

- **device** – **[in]** The identifier of the target device
- **minOffset** – **[out]** The retrieved MemClk VF min offset value
- **maxOffset** – **[out]** The retrieved MemClk VF max offset value

Returns

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

nvmlReturn_t **nvmlDeviceGetMemClkVfOffset**(
    nvmlDevice_t device, int *offset,
)

Retrieve the MemClk (Memory Clock) VF offset value.

Parameters

- **device** – **[in]** The identifier of the target device
- **offset** – **[out]** The retrieved MemClk VF offset value

Returns

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

nvmlReturn_t **nvmlDeviceGetMemoryBusWidth**(
nvmlDevice_t device, unsigned int *busWidth,
)

Gets the device's memory bus width.

Parameters

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

Returns

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

nvmlReturn_t **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 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.

Note

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

Parameters

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

Returns

- *NVML_SUCCESS* if *count* has been populated
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device*, *bitTyp*,e *counterType* or *location-Type* 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

nvmlReturn_t **nvmlDeviceGetMemoryInfo**(
    nvmlDevice_t device, *nvmlMemory_t* *memory,
)

Retrieves the amount of used, free, reserved and total memory available on the device, in bytes.

The reserved amount is supported on version 2 only.

For all products.

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

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

See *nvmlMemory_v2_t* for details on available memory info.

Note

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

Note

nvmlDeviceGetMemoryInfo_v2 adds additional memory information.

Note

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

Note

On certain SOC platforms, the integrated GPU (iGPU) does not use a dedicated framebuffer but instead shares memory with the system. As a result, *NVML_ERROR_NOT_SUPPORTED* will be returned in this case.

Note

In the MPS locality-domain view, this API returns MPS-provided total, used, and free memory for the locality domain. Reserved memory is included in used memory. See *MPS Locality Domain Device Enumeration*.

Parameters

- **`device`** – The identifier of the target device
- **`memory`** – Reference in which to return the memory information

Returns

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

nvmlReturn_t **nvmlDeviceGetMemoryInfo_v2**(
    nvmlDevice_t device, *nvmlMemory_v2_t* *memory,
)

Retrieves the amount of used, free, reserved and total memory available on the device, in bytes.

nvmlDeviceGetMemoryInfo_v2 accounts separately for reserved memory and includes it in the used memory amount.

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.

Note

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

Note

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

Note

On certain SOC platforms, the integrated GPU (iGPU) does not use a dedicated framebuffer but instead shares memory with the system. As a result, *NVML_ERROR_NOT_SUPPORTED* will be returned in this case.

Note

In the MPS locality-domain view, this API returns MPS-provided total, used, free, and reserved memory for the locality domain. See *MPS Locality Domain Device Enumeration*.

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetMemoryLimits_v1**(
nvmlDevice_t device, *nvmlGetMemoryLimits_v1_t* *limits,
)

Get the memory limits of the device for the cgroup partition.

This method will get the current memory limits of the device for the specified cgroup partition, as well as the current memory used against the limits.

For all products. For Linux only.

Note

MIG handles are not supported

Parameters

- **device** – **[in]** The identifier of the target device
- **limits** – **[inout]** A pointer to *nvmlGetMemoryLimits_v1_t*

Returns

- *NVML_SUCCESS* if the operation was successful
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *limits* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature
- *NVML_ERROR_NOT_FOUND* if the limits were not found for this device and cgroup
- *NVML_ERROR_OPERATING_SYSTEM* if the cgroup path cannot be opened
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

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

Parameters

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

Returns

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

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

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

For all cuda-capable discrete products with fans

Parameters

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

Returns

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

nvmlReturn_t **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

nvmlReturn_t **nvmlDeviceGetModuleId**(
    nvmlDevice_t device, unsigned int *moduleId,
)

Get a unique identifier for the device module on the baseboard.

This API retrieves a unique identifier for each GPU module that exists on a given baseboard. For non-baseboard products, this ID would always be 0.

Parameters

- **device** – The identifier of the target device
- **moduleId** – Unique identifier for the GPU module

Returns

- *NVML_SUCCESS* if *moduleId* has been successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *moduleId* is invalid
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **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 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

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

Retrieves the name of this device.

For all products.

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

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

Note

In the MPS locality-domain view, this API returns the name supplied by MPS for the locality domain. See *MPS Locality Domain Device Enumeration*.

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

nvmlReturn_t **nvmlDeviceGetNumFans**(
nvmlDevice_t device, unsigned int *numFans,
)

Retrieves the number of fans on the device.

For all discrete products with dedicated fans.

Parameters

- **device** – The identifier of the target device
- **numFans** – The number of fans

Returns

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

nvmlReturn_t **nvmlDeviceGetNumGpuCores**(
    nvmlDevice_t device, unsigned int *numCores,
)

Gets the device's core count.

Note

On MIG-enabled GPUs, querying the device's core count is currently not supported using this API. Please use *nvmlDeviceGetGpuInstanceProfileInfo* to fetch the MIG device's core count.

Parameters

- **device** – The identifier of the target device
- **numCores** – The number of cores for the specified device

Returns

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

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

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

For Turing or newer fully supported devices.

Note

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

Parameters

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

Returns

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

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

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

Note

In the MPS locality-domain view, this API returns the directed P2P status supplied by MPS for the two locality-domain handles. See *MPS Locality Domain Device Enumeration*.

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetPciInfoExt**(
nvmlDevice_t device, *nvmlPciInfoExt_t* *pci,
)

Retrieves PCI attributes of this device.

For all products.

See *nvmlPciInfoExt_v1_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

nvmlReturn_t **nvmlDeviceGetPciInfo_v3**(
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.

Note

In the MPS locality-domain view, this API and its v1 and v2 variants return the PCI attributes supplied by MPS. Multiple locality domains can report the same physical parent's PCI attributes. See *MPS Locality Domain Device Enumeration*.

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

nvmlReturn_t **nvmlDeviceGetPcieLinkMaxSpeed**(
    nvmlDevice_t device, unsigned int *maxSpeed,
)

Gets the device's PCIE Max Link speed in MBPS.

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetPcieReplayCounter**(
    nvmlDevice_t device, unsigned int *value,
)

Retrieve the PCIe replay counter.

For Kepler 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* has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, or *value* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature

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

nvmlReturn_t **nvmlDeviceGetPcieSpeed**(
    nvmlDevice_t device, unsigned int *pcieSpeed,
)

Gets the device's PCIe Link speed in Mbps.

Parameters

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

Returns

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

nvmlReturn_t **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 or newer fully supported devices.

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

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

nvmlReturn_t **nvmlDeviceGetPdi**(*nvmlDevice_t* device, *nvmlPdi_t* *pdi)

Retrieves the Per Device Identifier (PDI) associated with this device.

For Pascal or newer fully supported devices.

See *nvmlPdi_v1_t* for more information on the struct.

Parameters

- **device** – **[in]** The identifier of the target device
- **pdi** – **[out]** Reference to the caller-provided structure to return the GPU PDI

Returns

- *NVML_SUCCESS* if *pdi* has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, or *pdi* is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the version is invalid/unsupported
- *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

nvmlReturn_t **nvmlDeviceGetPerformanceModes**(
nvmlDevice_t device, *nvmlDevicePerfModes_t* *perfModes,
)

Retrieves a performance mode string with all the performance modes defined for this device along with their associated GPU Clock and Memory Clock values.

Not all tokens will be reported on all GPUs, and additional tokens may be added in the future. For backwards compatibility we still provide nvclock and memclock; those are the same as nvclockmin and memclockmin.

Note: These clock values take into account the offset set by clients through /ref nvmlDeviceSetClockOffsets.

Maximum available Pstate (P15) shows the minimum performance level (0) and vice versa.

Each performance modes are returned as a comma-separated list of “token=value” pairs. Each set of performance mode tokens are separated by a “;”. Valid tokens:

Token Value “perf” unsigned int - the Performance level “nvclock” unsigned int - the GPU clocks (in MHz) for the perf level “nvclockmin” unsigned int - the GPU clocks min (in MHz) for the perf level “nvclockmax” unsigned int - the GPU clocks max (in MHz) for the perf level “nvclockeditable” unsigned int - if the GPU clock domain is editable for the perf level “memclock” unsigned int - the memory clocks (in MHz) for the perf level “memclockmin” unsigned int - the memory clocks min (in MHz) for the perf level “memclockmax” unsigned int - the memory clocks max (in MHz) for the perf level “memclockeditable” unsigned int - if the memory clock domain is editable for the perf level “memtransferrate” unsigned int - the memory transfer rate (in MHz) for the perf level “memtransferratemin” unsigned int - the memory transfer rate min (in MHz) for the perf level “memtransferratemax” unsigned int - the memory transfer rate max (in MHz) for the perf

level “memtransferrateeditable” unsigned int - if the memory transfer rate is editable for the perf level

Example:

perf=0, nvclock=324, nvclockmin=324, nvclockmax=324, nvclockeditable=0, memclock=324, memclockmin=324, memclockmax=324, memclockeditable=0, memtransferrate=648, memtransferratemin=648, memtransferratemax=648, memtransferrateeditable=0 ; perf=1, nvclock=324, nvclockmin=324, nvclockmax=640, nvclockeditable=0, memclock=810, memclockmin=810, memclockmax=810, memclockeditable=0, memtransferrate=1620, memtransferrate=1620, memtransferrate=1620, memtransferrateeditable=0 ;

Parameters

- **device** – The identifier of the target device
- **perfModes** – Reference in which to return the performance level string

Returns

- *NVML_SUCCESS* if *perfModes* 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

nvmlReturn_t **nvmlDeviceGetPerformanceState**(
nvmlDevice_t device, *nvmlPstates_t* *pState,
)

Retrieves the current performance state for the device.

For Fermi 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

nvmlReturn_t **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.

See also

nvmlDeviceSetPersistenceMode()

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

nvmlReturn_t **nvmlDeviceGetPlatformInfo**(
    nvmlDevice_t device, *nvmlPlatformInfo_t* *platformInfo,
)

Get platform information of this device.

For Blackwell or newer fully supported devices.

See *nvmlPlatformInfo_v2_t* for more information on the struct.

Parameters

- **device** – The identifier of the target device
- **platformInfo** – Pointer to the caller-provided structure of nvmlPlatformInfo_t.

Returns

- *NVML_SUCCESS* If *platformInfo* has been retrieved

- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid or *platformInfo* is NULL
- *NVML_ERROR_NOT_SUPPORTED* If the device does not support this feature
- *NVML_ERROR_MEMORY* if system memory is insufficient
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *nvmlPlatformInfo_t* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **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 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

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

Retrieves the power management limit associated with this device.

For Fermi or newer fully supported devices.

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

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

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

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

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

For Kepler or newer fully supported devices.

See also

nvmlDeviceSetPowerManagementLimit

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

nvmlReturn_t **nvmlDeviceGetPowerManagementMode**(
nvmlDevice_t device, *nvmlEnableState_t* *mode,
)

Deprecated:

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

nvmlReturn_t **nvmlDeviceGetPowerMizerMode_v1**(
    nvmlDevice_t device, *nvmlDevicePowerMizerModes_v1_t* *powerMizerMode,
)

Retrieves current power mizer mode on this device.

PowerMizerMode provides a hint to the driver as to how to manage the performance of the GPU.

For Maxwell or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **powerMizerMode** – Reference in which to return the power mizer mode

Returns

- *NVML_SUCCESS* if *powerMizerMode* has been populated
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *powerMizerMode* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support powerMizerMode 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

nvmlReturn_t **nvmlDeviceGetPowerSource**(
    nvmlDevice_t device, *nvmlPowerSource_t* *powerSource,
)

Gets the devices power source.

Parameters

- **device** – The identifier of the target device
- **powerSource** – The power source of the device

Returns

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

nvmlReturn_t **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 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

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

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

memory)

For Fermi or newer fully supported devices.

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

See *NVML_FI_DEV_POWER_AVERAGE* and *NVML_FI_DEV_POWER_INSTANT* to query specific power values.

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

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

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

Retrieves the current utilization and process ID.

For Maxwell or newer fully supported devices.

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

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

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

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

Note

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

Parameters

- **`device`** – The identifier of the target device
- **`utilization`** – Pointer to caller-supplied buffer in which guest process utilization samples are returned
- **`processSamplesCount`** – Pointer to caller-supplied array size, and returns number of processes running
- **`lastSeenTimeStamp`** – Return only samples with timestamp greater than lastSeenTimeStamp.

Returns

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

nvmlReturn_t **`nvmlDeviceGetProcessesUtilizationInfo`**(
    nvmlDevice_t device, *nvmlProcessesUtilizationInfo_t* *procesesUtilInfo,
)

Retrieves the recent utilization and process ID for all running processes.

For Maxwell or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder, jpeg decoder, OFA (Optical Flow Accelerator) for all running processes. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by *procesesUtilInfo->procUtilArray*. One utilization sample structure is returned per process running, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values.

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

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

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

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

Note

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

Parameters

- **`device`** – The identifier of the target device
- **`procesesUtilInfo`** – Pointer to the caller-provided structure of nvmlProcessesUtilizationInfo_t.

Returns

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

- *NVML_ERROR_UNKNOWN* On any unexpected error

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

Get number of remapped rows.

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

For Ampere or newer fully supported devices.

Note

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

Parameters

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

Returns

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

nvmlReturn_t **nvmlDeviceGetRemappedRows_v2**(
nvmlDevice_t device, *nvmlRemappedRowsInfo_v2_t* *info,
)

Get the status of row remapper.

For Ampere or newer fully supported devices.

Note

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

Parameters

- **device** – The identifier of the target device
- **info** – Reference for *nvmlRemappedRowsInfo_v2_t*

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_INVALID_ARGUMENT* If *info* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or if the device doesn't support this feature
- *NVML_ERROR_UNKNOWN* Unexpected error

nvmlReturn_t **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 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

nvmlReturn_t **nvmlDeviceGetRetiredPagesPendingStatus**(
nvmlDevice_t device, *nvmlEnableState_t* *isPending,
)

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

For Kepler 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

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

Returns the list of retired pages by source, including pages that are pending retirement The address information provided from this API is the hardware address of the page that was retired.

Note that this does not match the virtual address used in CUDA, but will match the address information in Xid 63

For Kepler or newer fully supported devices.

Note

nvmlDeviceGetRetiredPages_v2 adds an additional timestamps parameter to return the time of each page's retirement. This is supported for Pascal and newer architecture.

Parameters

- **device** – The identifier of the target device
- **cause** – Filter page addresses by cause of retirement

- **pageCount** – Reference in which to provide the *addresses* buffer size, and to return the number of retired pages that match *cause* Set to 0 to query the size without allocating an *addresses* buffer
- **addresses** – Buffer to write the page addresses into
- **timestamps** – Buffer to write the timestamps of page retirement, additional for _v2

Returns

- *NVML_SUCCESS* if *pageCount* was populated and *addresses* was filled
- *NVML_ERROR_INSUFFICIENT_SIZE* if *pageCount* indicates the buffer is not large enough to store all the matching page addresses. *pageCount* is set to the needed size.
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, *pageCount* is NULL, *cause* is invalid, or *addresses* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if the device doesn't support this feature
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetRowRemapperHistogram**(
    nvmlDevice_t device, *nvmlRowRemapperHistogramValues_t* *values,
)

Get the row remapper histogram.

Returns the remap availability for each bank on the GPU.

Parameters

- **device** – Device handle
- **values** – Histogram values

Returns

- *NVML_SUCCESS* On success
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetRunningProcessDetailList**(
    nvmlDevice_t device, *nvmlProcessDetailList_t* *plist,
)

Get information about running processes on a device for input context.

For Hopper or newer fully supported devices.

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

To determine the size of the *plist->procArray* array to allocate, call the function with *plist->numProcArrayEntries* set to zero and *plist->procArray* set to NULL. The return code will be either NVML_ERROR_INSUFFICIENT_SIZE (if there are valid processes of type *plist->mode* to report on,

in which case the *plist->numProcArrayEntries* field will indicate the required number of entries in the array) or NVML_SUCCESS (if no processes of type *plist->mode* exist).

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

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

Note

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

Parameters

- **device** – The device handle or MIG device handle
- **plist** – Reference in which to process detail list *plist->version* The api version *plist->mode* The process mode *plist->procArray* Reference in which to return the process information *plist->numProcArrayEntries* Proc array size of returned entries

Returns

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

nvmlReturn_t **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 or newer fully supported devices.

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

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

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

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

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

Note

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

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

nvmlReturn_t **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 inforom.

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

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

nvmlReturn_t **nvmlDeviceGetSramEccErrorStatus**(
    nvmlDevice_t device, *nvmlEccSramErrorStatus_t* *status,
)

Get SRAM ECC error status of this device.

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

See *nvmlEccSramErrorStatus_v1_t* for more information on the struct.

Parameters

- **device** – The identifier of the target device
- **status** – Returns SRAM ECC error status

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 *counters* is NULL
- *NVML_ERROR_NOT_SUPPORTED* If the device does not support this feature

- *NVML_ERROR_GPU_IS_LOST* If the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *nvmlEccSramErrorStatus_t* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetSramUniqueUncorrectedEccErrorCounts**(
nvmlDevice_t device, *nvmlEccSramUniqueUncorrectedErrorCounts_t* *errorCounts,
)

Retrieves the counts of SRAM unique uncorrected ECC errors.

For Blackwell or newer fully supported devices.

Reads SRAM unique uncorrected ECC error counts. The total number of unique errors is returned by *errorCounts->entryCount*. Error counts are returned as an array of in the caller-supplied buffer pointed at by *errorCounts->entries*. Each error count entry holds the location/address of the unique error, the error count and whether the error is parity or not.

To read SRAM unique uncorrected ECC error counts, first determine the size of buffer required to hold the error counts by invoking the function with *errorCounts->entries* set to NULL. The required array size is returned in *errorCounts->entryCount* . The caller should allocate a buffer of size “errorCounts->entryCount *

sizeof(nvmlEccSramUniqueUncorrectedErrorCounts_t)”. Invoke the function again with the allocated buffer passed in *errorCounts->entries*. This time *errorCounts->entryCount* will be taken as the entry array size that caller allocates for *errorCounts->entries*.

On successful return of the second query, the function updates *errorCounts->entries* with all unique errors. This may fail if *errorCounts->entryCount* is smaller than the actual number of unique errors. This can happen in cases like new errors occur since the previous query of *errorCounts->entryCount*. No matter the query succeeds or not, the latest number of unique errors will be returned in *errorCounts->entryCount*.

Note

The query is only supported when ECC mode is enabled.

Parameters

- **device** – The identifier of the target device
- **errorCounts** – Pointer to caller-supplied array which returns the unique error count entries

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, *errorCounts->entryCount* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature or ECC mods is not enabled

- *NVML_ERROR_INSUFFICIENT_SIZE* if the allocated error entry array is not big enough
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

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

For all fully supported products.

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

See also

NvmlClocksEventReasons

See also

nvmlDeviceGetCurrentClocksEventReasons

Parameters

- **device** – The identifier of the target device
- **supportedClocksEventReasons** – Reference in which to return bitmask of supported clocks event reasons

Returns

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

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

Deprecated:

Use *nvmlDeviceGetSupportedClocksEventReasons* instead

nvmlReturn_t **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 *nvmlDeviceSetGpuLockedClocks*.

For Kepler or newer fully supported devices.

See also

nvmlDeviceSetGpuLockedClocks

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

nvmlReturn_t **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 *nvmlDeviceSetMemoryLockedClocks*.

For Kepler or newer fully supported devices.

See also

nvmlDeviceSetMemoryLockedClocks

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

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

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

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

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

Parameters

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

Returns

- *NVML_SUCCESS* if *pstates* array has been retrieved
- *NVML_ERROR_INSUFFICIENT_SIZE* if the the container supplied was not large enough to hold the resulting list
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized

- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *pstates* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support performance state readings
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

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

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

For all discrete products with dedicated fans.

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

Parameters

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

Returns

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

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

Deprecated:

Use *nvmlDeviceGetTemperatureV* instead

nvmlReturn_t **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 or newer fully supported devices.

See *nvmlTemperatureThresholds_t* for details on available temperature thresholds.

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

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

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

nvmlReturn_t **nvmlDeviceGetTemperatureV**(
    nvmlDevice_t device, *nvmlTemperature_t* *temperature,
)

Retrieves the current temperature readings (in degrees C) for the given device.

For all products.

Note: This API reports the current GPU temperature rounded to a whole degree Celsius. For the current GPU temperature at higher precision, use *nvmlDeviceGetFieldValues* with the *NVML_FI_DEV_GPU_TEMP* field, which returns the temperature in degrees C as a double value.

Parameters

- **device** – **[in]** Target device identifier.
- **temperature** – **[inout]** Structure specifying the sensor type (input) and retrieved temperature value (output).

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

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

Used to execute a list of thermal system instructions.

Parameters

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

Returns

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

nvmlReturn_t **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

nvmlReturn_t **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

nvmlReturn_t **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.

See also

nvmlDeviceClearEccErrorCounts()

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

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

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

For Volta or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **energy** – Reference in which to return the energy consumption information

Returns

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

nvmlReturn_t **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 96 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_UUID_V2_BUFFER_SIZE.

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

Note

In the MPS locality-domain view, this API returns the UUID supplied by MPS for the locality domain. See *MPS Locality Domain Device Enumeration*.

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

nvmlReturn_t **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.

Note

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

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

nvmlReturn_t **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

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

Deprecated:

Use *nvmlDeviceGetFieldValues* to query this data. This API will be removed in CUDA 14.0.

Translations are as follows:

NVML_PERF_POLICY_POWER -> NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_POWER_CAP
NVML_PERF_POLICY_THERMAL -> NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_THERM_SLOWDOWN
NVML_PERF_POLICY_SYNC_BOOST -> NVML_FI_DEV_CLOCKS_EVENT_REASON_SYNC_BOOST
NVML_PERF_POLICY_BOARD_LIMIT -> NVML_FI_DEV_PERF_POLICY_BOARD_LIMIT
NVML_PERF_POLICY_LOW_UTILIZATION -> NVML_FI_DEV_PERF_POLICY_LOW_UTILIZATION
NVML_PERF_POLICY_RELIABILITY -> NVML_FI_DEV_PERF_POLICY_RELIABILITY
NVML_PERF_POLICY_TOTAL_APP_CLOCKS -> DEPRECATED, Do not use
NVML_PERF_POLICY_TOTAL_BASE_CLOCKS -> NVML_FI_DEV_PERF_POLICY_TOTAL_BASE_CLOCKS

nvmlReturn_t **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

nvmlReturn_t **nvmlDevicePerfMetricsGetSamples_v1**(
    nvmlDevice_t device, *nvmlPerfMetricsSamples_v1_t* *samples,
)

Get Performance Metric samples.

See *nvmlPerfMetricsSamples_v1_t* for more information on the struct.

Parameters

- **device** – **[in]** The identifier of the target device
- **samples** – **[out]** Reference to nvmlPerfMetricsSamples_v1_t.

Returns

- *NVML_SUCCESS* if the query is successful
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device

nvmlReturn_t **nvmlDeviceSetAdaptiveTgpMode_v1**(
nvmlDevice_t device, *nvmlEnableState_t* mode,
)

Request to enable or disable Adaptive TGP Mode for a GPU.

RUBIN_OR_NEWER% Requires root/admin privileges.

Adaptive TGP Mode assigns tailored power budgets to two binned GPU parts within the same module, reducing node-to-node and rack-to-rack performance variation. An out-of-band administrator policy may override the in-band request; use *nvmlDeviceGetAdaptiveTgpModeInfo_v1* to query the arbitrated state.

See also

nvmlDeviceGetAdaptiveTgpModeInfo_v1()

Parameters

- **device** – The identifier of the target device
- **mode** – NVML_FEATURE_ENABLED or NVML_FEATURE_DISABLED

Returns

- *NVML_SUCCESS* if the request was accepted
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *mode* is not a valid *nvmlEnableState_t*
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support Adaptive TGP Mode
- *NVML_ERROR_NO_PERMISSION* if the caller lacks root/admin privileges
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceSetClockOffsets**(
    nvmlDevice_t device, *nvmlClockOffset_t* *info,
)

Control current clock offset of some clock domain for a given PState.

For Maxwell or newer fully supported devices.

Requires privileged user.

Parameters

- **device** – The identifier of the target device
- **info** – Structure specifying the clock type (input), the pstate (input) and clock offset value (input)

Returns

- *NVML_SUCCESS* If everything worked
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_NO_PERMISSION* If the user doesn't have permission to perform this operation
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *type* or *pstate* are invalid or both *clockOffsetMHz* is out of allowed range.
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the provided version is invalid/unsupported
- *NVML_ERROR_NOT_SUPPORTED* If the device does not support this feature

nvmlReturn_t **nvmlDeviceSetConfComputeUnprotectedMemSize**(
    nvmlDevice_t device, unsigned long long sizeKiB,
)

Set Conf Computing Unprotected Memory Size.

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

Parameters

- **device** – Device Handle
- **sizeKiB** – Unprotected Memory size to be set in KiB

Returns

- *NVML_SUCCESS* if *sizeKiB* successfully set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device

nvmlReturn_t **nvmlDeviceSetDramEncryptionMode**(
    nvmlDevice_t device, const *nvmlDramEncryptionInfo_t* *dramEncryption,
)

Set the DRAM Encryption mode for the device.

For Kepler or newer fully supported devices. Only applicable to devices that support DRAM Encryption. Requires *NVML_INFOROM_DEN* version 1.0 or higher. Requires root/admin permissions.

The DRAM Encryption mode determines whether the GPU enables its DRAM Encryption support.

This operation takes effect after the next reboot.

See *nvmlEnableState_t* for details on available modes.

See also

nvmlDeviceGetDramEncryptionMode()

Parameters

- **device** – The identifier of the target device
- **dramEncryption** – The target DRAM Encryption mode

Returns

- *NVML_SUCCESS* if the DRAM Encryption mode was set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *DRAM* Encryption 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_ARGUMENT_VERSION_MISMATCH* if the argument version is not supported
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceSetHostname_v1**(
    nvmlDevice_t device, *nvmlHostname_v1_t* *hostname,
)

Set the hostname for the device.

For Blackwell or newer fully supported devices. Requires root/admin permissions. Supported on Linux only.

Sets a hostname string for the GPU device. This operation takes effect immediately.

The hostname is not stored persistently across GPU resets or driver reloads.

See also

nvmlDeviceGetHostname_v1()

Parameters

- **device** – The identifier of the target device
- **hostname** – Reference to the caller-provided *nvmlHostname_v1_t* struct containing the hostname

Returns

- *NVML_SUCCESS* if the hostname was set successfully
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *hostname* is NULL or contains invalid characters
- *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

nvmlReturn_t **nvmlDeviceSetMemoryLimits_v1**(
    nvmlDevice_t device, *nvmlSetMemoryLimits_v1_t* *limits,
)

Set the memory limits of the device for the cgroup partition.

This method will set the memory limits of the device for the specified cgroup partition. The limits will indicate the amount of memory that can be allocated for the device for use of an application in that cgroup.

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

Note

MIG handles are not supported

Parameters

- **device** – **[in]** The identifier of the target device
- **limits** – **[in]** A pointer to *nvmlSetMemoryLimits_v1_t* where the limits can be set

Returns

- *NVML_SUCCESS* if the operation was successful

- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, *limits* is NULL, the softLimit exceeds the hardLimit, or a limit exceeds total device memory
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature
- *NVML_ERROR_OPERATING_SYSTEM* if the cgroup path cannot be opened
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceSetPowerManagementLimit_v2**(
nvmlDevice_t device, *nvmlPowerValue_v2_t* *powerValue,
)

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.

See *nvmlPowerValue_v2_t* for more information on the struct.

This API replaces nvmlDeviceSetPowerManagementLimit. It can be used as a drop-in replacement for the older version.

See also

NVML_FI_DEV_POWER_AVERAGE

See also

NVML_FI_DEV_POWER_INSTANT

See also

NVML_FI_DEV_POWER_MIN_LIMIT

See also

NVML_FI_DEV_POWER_MAX_LIMIT

See also

NVML_FI_DEV_POWER_CURRENT_LIMIT

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
- **powerValue** – 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 *powerValue* is NULL or contains invalid values
- *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

nvmlReturn_t **nvmlDeviceSetPowerMizerMode_v1**(
    nvmlDevice_t device, *nvmlDevicePowerMizerModes_v1_t* *powerMizerMode,
)

Sets the new power mizer mode.

For Maxwell or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **powerMizerMode** – Reference in which to set the power mizer mode.

Returns

- *NVML_SUCCESS* if *powerMizerMode* has been populated
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *powerMizerMode* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support powerMizerMode 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

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

nvmlReturn_t **nvmlSystemGetConfComputeCapabilities**(
nvmlConfComputeSystemCaps_t *capabilities,
)

Get Conf Computing System capabilities.

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

Parameters

capabilities – System CC capabilities

Returns

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

nvmlReturn_t **nvmlSystemGetConfComputeGpusReadyState**(
unsigned int *isAcceptingWork,
)

Get Conf Computing GPUs ready state.

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

return

- *NVML_SUCCESS* if *current* GPUs ready state were successfully queried
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *isAcceptingWork* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device

Parameters

isAcceptingWork – Returns GPU current work accepting state, NVML_CC_ACCEPTING_CLIENT_REQUESTS_TRUE or NVML_CC_ACCEPTING_CLIENT_REQUESTS_FALSE

nvmlReturn_t **nvmlSystemGetConfComputeKeyRotationThresholdInfo**(
nvmlConfComputeGetKeyRotationThresholdInfo_t *pKeyRotationThrInfo,
)

Get Conf Computing key rotation threshold detail.

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

Parameters

pKeyRotationThrInfo – Reference in which to return the key rotation threshold data

Returns

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

nvmlReturn_t **nvmlSystemGetConfComputeSettings**(
nvmlSystemConfComputeSettings_t *settings,
)

Get Conf Computing System Settings.

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

Parameters

settings – System CC settings

Returns

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

nvmlReturn_t **nvmlSystemGetConfComputeSettings_v2**(
nvmlSystemConfComputeSettings_v2_t *settings,
)

Get Conf Computing System Settings, version 2.

Returns everything *nvmlSystemGetConfComputeSettings* reports, plus the TDISP state of the GPUs in *tdispState*.

All GPUs, and therefore all TDIs, are expected to be in the same state. If they are not, the system level state is ambiguous and NVML_ERROR_UNKNOWN is returned.

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

Parameters

settings – System CC settings

Returns

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

nvmlReturn_t **nvmlSystemGetConfComputeState**(
nvmlConfComputeSystemState_t *state,
)

Get Conf Computing System State.

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

Parameters

state – System CC State

Returns

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

nvmlReturn_t **nvmlSystemSetConfComputeGpusReadyState**(
unsigned int isAcceptingWork,
)

Set Conf Computing GPUs ready state.

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

return

- *NVML_SUCCESS* if *current* GPUs ready state is successfully set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized

- *NVML_ERROR_INVALID_ARGUMENT* if *isAcceptingWork* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device

Parameters

isAcceptingWork – GPU accepting new work, NVML_CC_ACCEPTING_CLIENT_REQUESTS_TRUE or NVML_CC_ACCEPTING_CLIENT_REQUESTS_FALSE

nvmlReturn_t **nvmlSystemSetConfComputeKeyRotationThresholdInfo**(
nvmlConfComputeSetKeyRotationThresholdInfo_t *pKeyRotationThrInfo,
)

Set Conf Computing key rotation threshold.

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

This function is to set the confidential compute key rotation threshold parameters. *pKeyRotationThrInfo->maxAttackerAdvantage* should be in the range from NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MIN to NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MAX. Default value is 60.

Parameters

pKeyRotationThrInfo – Reference to the key rotation threshold data

Returns

- *NVML_SUCCESS* if *key* rotation threshold max attacker advantage has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *memory* is NULL
- *NVML_ERROR_INVALID_STATE* if confidential compute GPU ready state is enabled
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device
- *NVML_ERROR_UNKNOWN* on any unexpected error

5.2.1.7.9 Typedefs

typedef *nvmlTemperature_v1_t* **nvmlTemperature_t**

5.2.1.8 Device Structs

5.2.1.8.1 Macros

NVML_DEVICE_MEMORY_LIMIT_MAX
Value for "maximum" memory limit.

NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE
Buffer size guaranteed to be large enough for pci bus id.

NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE
Buffer size guaranteed to be large enough for pci bus id for `busIdLegacy` .

NVML_DEVICE_PCI_BUS_ID_FMT
PCI format string for `busId` .

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

NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT
PCI format string for `busIdLegacy` .

NVML_DEVICE_UUID_ASCII_LEN
UUID length in ASCII format.

NVML_DEVICE_UUID_BINARY_LEN
UUID length in binary format.

NVML_MAX_PHYSICAL_BRIDGE
Maximum limit on Physical Bridges per Board.

NVML_MAX_THERMAL_SENSORS_PER_GPU
Maximum number of thermal sensors per GPU.

NVML_NVLINK_MAX_LINKS
Maximum number of NvLink links supported.

NVML_PERF_CF_PM_SENSOR_MAX_SIGNALS
Maximum number of BA PM sensor signals.

NVML_PERF_METRICS_CONTROLLER_DLPPC_2X_PWR_POLICY_RELATIONSHIP_SET_LIMITS_MAX
Maximum number of power policy relationship set limits for DLPPC 2x controller.

NVML_PERF_METRICS_CONTROLLER_SAMPLE_CONTROLLER_MAX_NUM
Maximum number of controllers that can be sampled.

NVML_PERF_METRICS_CONTROLLER_STATUS_DLPPC_2X_DRAMCLK_NUM
Number of DRAMCLK frequencies tracked by DLPPC 2x controller status.

NVML_PERF_METRICS_CONTROLLER_TYPE_DLPPC_2X
Controller type identifier for DLPPC 2x.

NVML_PERF_METRICS_CONTROLLER_TYPE_PFPP_1X
Controller type identifier for PFPP 1x.

NVML_PERF_METRICS_NNE_DESC_INFERENCE_LOOPS_MAX
Maximum number of NNE descriptor inference loops.

NVML_PERF_METRICS_PWR_MODEL_DLPPM_1X_MAX_CORE_RAILS
Maximum number of core rails for DLPPM 1x power model.

NVML_PERF_METRICS_PWR_MODEL_METRICS_DLPPM_1X_OBESRVED_INTIAL_DRAMCLK_ESTIMATES_MAX
Maximum number of initial DRAMCLK estimates for DLPPM 1x observed metrics.

NVML_PERF_METRICS_PWR_MODEL_SCALE_LOOPS_MAX_PFPP_1X
Maximum number of power model scale loops for PFPP 1x.

NVML_PERF_METRICS_PWR_MODEL_SCALE_METRICS_INPUT_MAX
Maximum number of power model scale metrics inputs.

NVML_PERF_METRICS_PWR_MODEL_SCALE_METRICS_PFPP_1X_GPCCLK_IDX
Index for GPCCLK frequency in PFPP 1x scale metrics.

NVML_PERF_METRICS_SAMPLE_COUNT
Total number of performance metrics samples that can be collected.

NVML_TOPOLOGY_CPU
Topology level for node.

NVML_VALUE_NOT_AVAILABLE
Special constant that some fields take when they are not available.

nvmlC2cModeInfo_v1
Version macro for *nvmlC2cModeInfo_v1_t* .

nvmlCoolerInfo_v1
Version macro for *nvmlCoolerInfo_v1_t* .

nvmlDeviceAddressingMode_v1
Version macro for *nvmlDeviceAddressingMode_v1_t* .

nvmlMemory_v2
Version macro for *nvmlMemory_v2_t* .

nvmlPciInfoExt_v1
Version macro for *nvmlPciInfoExt_v1_t* .

nvmlPdi_v1
Version macro for *nvmlPdi_v1_t* .

nvmlProcessDetailList_v1
nvmlProcessDetailList version

nvmlRepairStatus_v1
Version macro for *nvmlRepairStatus_v1_t* .

nvmlUUID_v1
Version macro for *nvmlUUID_v1_t* .

5.2.1.8.2 Enumerations

nvmlBridgeChipType_t
Enum to represent type of bridge chip.

nvmlCoolerControl_t
Cooler control type.

nvmlCoolerTarget_t
Cooler's target.

nvmlDeviceAddressingModeType_t
Enum to represent device addressing mode values.

nvmlGpuP2PCapsIndex_t

nvmlGpuP2PStatus_t

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

nvmlIntNvLinkDeviceType_t
Enum to represent NvLink's remote device type.

nvmlNvLinkCapability_t
Enum to represent NvLink queryable capabilities.

nvmlNvLinkErrorCounter_t
Enum to represent NvLink queryable error counters.

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.

nvmlNvLinkUtilizationCountUnits_t
Enum to represent the NvLink utilization counter packet units.

nvmlPcieUtilCounter_t
Represents the queryable PCIe utilization counters.

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

nvmlProcessMode_t
Enum to represent process mode.

nvmlSamplingType_t
Represents Type of Sampling Event.

nvmlThermalController_t
Represents the thermal sensor controllers.

nvmlThermalTarget_t
Represents the thermal sensor targets.

nvmlUUIDType_t
Enum to represent different UUID types.

nvmlValueType_t
Represents the type for sample value returned.

5.2.1.8.3 Structs

nvmlBAR1Memory_t
BAR1 Memory allocation Information for a device.

nvmlBBXTimeData_v1_t
BBX Time Data.

nvmlBridgeChipHierarchy_t
This structure stores the complete Hierarchy of the Bridge Chip within the board.

nvmlBridgeChipInfo_t
Information about the Bridge Chip Firmware.

nvmlC2cModeInfo_v1_t
C2C Mode information for a device.

nvmlCoolerInfo_v1_t

nvmlCoreRailMetrics_t
Metrics for all core rails in the system.

nvmlDeviceAddressingMode_v1_t
Struct to represent device addressing mode information.

nvmlDeviceAttributes_t

nvmlEccBankRemapperHistogram_v1_t
Structure to store bank remapper histogram.

nvmlEccBankRemapperStatus_v1_t
Structure to store bank remapper status.

nvmlEccErrorCounts_t
Detailed ECC error counts for a device.

nvmlGetMemoryLimits_v1_t
Describes the current memory limits that are set for the device.

nvmlGpuThermalSettings_t
Struct to hold the thermal sensor settings.

nvmlGpuUptime_v1_t
GPU Uptime.

nvmlMemory_t
Memory allocation information for a device (v1).

nvmlMemory_v2_t
Memory allocation information for a device (v2).

nvmlNvLinkUtilizationControl_t
Struct to define the NVLINK counter controls.

nvmlObservedMetrics_t
Observed metrics from the power model, including initial DRAMCLK estimates and current measurements.

nvmlPciInfoExt_v1_t
PCI information about a GPU device.

nvmlPciInfo_t
PCI information about a GPU device.

nvmlPdi_v1_t
Struct to represent the NVML PDI information.

nvmlPerfMetricControllerSample_t
Performance metrics sample from a controller, which can be either DLPPC 2x or PFPP 1x.

nvmlPerfMetricsDlppc2xSample_t
Performance metrics sample for DLPPC 2x controller.

nvmlPerfMetricsPfpp1xSample_t
Performance metrics sample for PFPP 1x controller.

nvmlPerfMetricsSample_t
Single performance metrics sample containing data from one or more controllers.

nvmlPerfMetricsSamples_v1_t
Collection of performance metrics samples (version 1).

nvmlPmgrPwrTuple_t
Power tuple containing power consumption in milliwatts.

nvmlProcessDetailList_v1_t
Information about all running processes on the GPU for the given mode.

nvmlProcessDetail_v1_t
Information about running process on the GPU with protected memory.

nvmlProcessInfo_t
Information about running compute processes on the GPU.

nvmlProcessInfo_v1_t
Information about running compute processes on the GPU, legacy version for older versions of the API.

nvmlPwrModelMetricsDlppm1xDramclkEstimates_t
DRAMCLK estimates containing multiple estimated metrics for different DRAMCLK frequencies.

nvmlPwrModelMetricsDlppm1xPerf_t
Performance metrics for DLPPM 1x power model.

nvmlPwrModelMetricsDlppm1x_t
Complete power model metrics for DLPPM 1x, including rail metrics and TGP power.

nvmlPwrModelMetricsPfpp1x_t
Complete power model metrics for PFPP 1x, including estimated metrics and key operating points.

nvmlPwrModelMetricsSamplePfpp1x_t
Power model metrics sample for PFPP 1x, containing frequency inputs and estimated TGP.

nvmlPwrModelOperatingPointPfpp1x_t
Operating point for PFPP 1x power model, defining a frequency-power pair.

nvmlRailMetrics_t
Metrics for a single power rail, including frequency and utilization.

nvmlRepairStatus_v1_t
Struct to represent the NVML repair status.

nvmlRowRemapperHistogramValues_t
Possible values that classify the remap availability for each bank.

nvmlSample_t
Information for Sample.

nvmlSetMemoryLimits_v1_t
Describes the memory limits that can be set for the device.

nvmlUUID_v1_t
Struct to represent NVML UUID information.

nvmlUnrepairableMemoryStatus_v1_t
Struct to represent the NVML unrepairable memory status.

nvmlUtilization_t
Utilization information for a device.

nvmlViolationTime_t
Struct to hold perf policy violation status data.

5.2.1.8.4 Typedefs

nvmlCoolerInfo_t

nvmlDeviceAddressingMode_t

nvmlDevice_t

nvmlGpuInstance_t

nvmlPciInfoExt_t

nvmlPdi_t

nvmlProcessDetailList_t

nvmlRepairStatus_t

nvmlUUID_t

5.2.1.8.5 Unions

nvmlUUIDValue_t
Union to represent different UUID values.

nvmlValue_t
Union to represent different types of Value.

5.2.1.8.6 Macros

NVML_DEVICE_MEMORY_LIMIT_MAX
Value for “maximum” memory limit.

NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE
Buffer size guaranteed to be large enough for pci bus id.

Buffer size for PCI bus ID.

NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE
Buffer size guaranteed to be large enough for pci bus id for `busIdLegacy`.

Buffer size for legacy PCI bus ID.

NVML_DEVICE_PCI_BUS_ID_FMT
PCI format string for `busId`.

PCI bus ID format.

NVML_DEVICE_PCI_BUS_ID_FMT_ARGS(pciInfo)
Utility macro for filling the pci bus id format from a *nvmlPciInfo_t*.

Macro for formatting PCI bus ID arguments.

NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT
PCI format string for `busIdLegacy`.

Legacy PCI bus ID format.

NVML_DEVICE_UUID_ASCII_LEN

UUID length in ASCII format.

Length of UUID in ASCII format.

NVML_DEVICE_UUID_BINARY_LEN

UUID length in binary format.

Length of UUID in binary format.

NVML_MAX_PHYSICAL_BRIDGE

Maximum limit on Physical Bridges per Board.

Maximum number of physical bridges.

NVML_MAX_THERMAL_SENSORS_PER_GPU

Maximum number of thermal sensors per GPU.

NVML_NVLINK_MAX_LINKS

Maximum number of NvLink links supported.

Maximum number of NVLink links supported.

NVML_PERF_CF_PM_SENSOR_MAX_SIGNALS

Maximum number of BA PM sensor signals.

NVML_PERF_METRICS_CONTROLLER_DLPPC_2X_PWR_POLICY_RELATIONSHIP_SET_LIMITS_MAX

Maximum number of power policy relationship set limits for DLPPC 2x controller.

NVML_PERF_METRICS_CONTROLLER_SAMPLE_CONTROLLER_MAX_NUM

Maximum number of controllers that can be sampled.

NVML_PERF_METRICS_CONTROLLER_STATUS_DLPPC_2X_DRAMCLK_NUM

Number of DRAMCLK frequencies tracked by DLPPC 2x controller status.

NVML_PERF_METRICS_CONTROLLER_TYPE_DLPPC_2X

Controller type identifier for DLPPC 2x.

NVML_PERF_METRICS_CONTROLLER_TYPE_PFPP_1X

Controller type identifier for PFPP 1x.

NVML_PERF_METRICS_NNE_DESC_INFERENCE_LOOPS_MAX

Maximum number of NNE descriptor inference loops.

NVML_PERF_METRICS_PWR_MODEL_DLPPM_1X_MAX_CORE_RAILS

Maximum number of core rails for DLPPM 1x power model.

NVML_PERF_METRICS_PWR_MODEL_METRICS_DLPPM_1X_OBESRVED_INTIAL_DRAMCLK_ESTIMATES_MAX
Maximum number of initial DRAMCLK estimates for DLPPM 1x observed metrics.

NVML_PERF_METRICS_PWR_MODEL_SCALE_LOOPS_MAX_PFPP_1X
Maximum number of power model scale loops for PFPP 1x.

NVML_PERF_METRICS_PWR_MODEL_SCALE_METRICS_INPUT_MAX
Maximum number of power model scale metrics inputs.

NVML_PERF_METRICS_PWR_MODEL_SCALE_METRICS_PFPP_1X_GPCCLK_IDX
Index for GPCCLK frequency in PFPP 1x scale metrics.

NVML_PERF_METRICS_SAMPLE_COUNT
Total number of performance metrics samples that can be collected.

NVML_TOPOLOGY_CPU
Topology level for node.

NVML_VALUE_NOT_AVAILABLE
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. Macro for unavailable values.

nvmlC2cModeInfo_v1
Version macro for nvmlC2cModeInfo_v1_t.

nvmlCoolerInfo_v1
Version macro for *nvmlCoolerInfo_v1_t*.

nvmlDeviceAddressingMode_v1
Version macro for nvmlDeviceAddressingMode_v1_t.

nvmlMemory_v2
Version macro for nvmlMemory_v2_t.

nvmlPciInfoExt_v1
Version macro for nvmlPciInfoExt_v1_t.

nvmlPdi_v1
Version macro for nvmlPdi_v1_t.

nvmlProcessDetailList_v1
nvmlProcessDetailList version

Version macro for nvmlProcessDetailList_v1_t

nvmlRepairStatus_v1

Version macro for nvmlRepairStatus_v1_t.

nvmlUUID_v1

Version macro for nvmlUUID_v1_t.

5.2.1.8.7 Enumerations

enum **nvmlBridgeChipType_t**

Enum to represent type of bridge chip.

Values:

enumerator **NVML_BRIDGE_CHIP_PLX**

enumerator **NVML_BRIDGE_CHIP_BRO4**

enum **nvmlCoolerControl_t**

Cooler control type.

Values:

enumerator **NVML_THERMAL_COOLER_SIGNAL_NONE**

This cooler has no control signal.

enumerator **NVML_THERMAL_COOLER_SIGNAL_TOGGLE**

This cooler can only be toggled either ON or OFF (eg a switch).

enumerator **NVML_THERMAL_COOLER_SIGNAL_VARIABLE**

This cooler's level can be adjusted from some minimum to some maximum (eg a knob).

enumerator **NVML_THERMAL_COOLER_SIGNAL_COUNT**

enum **nvmlCoolerTarget_t**

Cooler's target.

Values:

enumerator **NVML_THERMAL_COOLER_TARGET_NONE**

This cooler cools nothing.

enumerator **NVML_THERMAL_COOLER_TARGET_GPU**

This cooler can cool the GPU.

enumerator **NVML_THERMAL_COOLER_TARGET_MEMORY**

This cooler can cool the memory.

enumerator **NVML_THERMAL_COOLER_TARGET_POWER_SUPPLY**

This cooler can cool the power supply.

enumerator **NVML_THERMAL_COOLER_TARGET_GPU_RELATED**

This cooler cools all of the components related to its target gpu. GPU_RELATED = GPU | MEMORY | POWER_SUPPLY.

enum **nvmlDeviceAddressingModeType_t**

Enum to represent device addressing mode values.

Values:

enumerator **NVML_DEVICE_ADDRESSING_MODE_NONE**

No active mode.

enumerator **NVML_DEVICE_ADDRESSING_MODE_HMM**

Heterogeneous Memory Management mode.

enumerator **NVML_DEVICE_ADDRESSING_MODE_ATS**

Address Translation Services mode.

enum **nvmlGpuP2PCapsIndex_t**

Values:

enumerator **NVML_P2P_CAPS_INDEX_READ**

enumerator **NVML_P2P_CAPS_INDEX_WRITE**

enumerator **NVML_P2P_CAPS_INDEX_NVLINK**

enumerator **NVML_P2P_CAPS_INDEX_ATOMICS**

enumerator **NVML_P2P_CAPS_INDEX_PCI**

enumerator **NVML_P2P_CAPS_INDEX_PROP**

enumerator **NVML_P2P_CAPS_INDEX_UNKNOWN**

enum **nvmlGpuP2PStatus_t**

Values:

enumerator **NVML_P2P_STATUS_OK**

enumerator **NVML_P2P_STATUS_CHIPSET_NOT_SUPPORED**

enumerator **NVML_P2P_STATUS_CHIPSET_NOT_SUPPORTED**

enumerator **NVML_P2P_STATUS_GPU_NOT_SUPPORTED**

enumerator **NVML_P2P_STATUS_IOH_TOPOLOGY_NOT_SUPPORTED**

enumerator **NVML_P2P_STATUS_DISABLED_BY_REGKEY**

enumerator **NVML_P2P_STATUS_NOT_SUPPORTED**

enumerator **NVML_P2P_STATUS_UNKNOWN**

enum **nvmlGpuTopologyLevel_t**

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

Values:

enumerator **NVML_TOPOLOGY_INTERNAL**

enumerator **NVML_TOPOLOGY_SINGLE**

enumerator **NVML_TOPOLOGY_MULTIPLE**

enumerator **NVML_TOPOLOGY_HOSTBRIDGE**

enumerator **NVML_TOPOLOGY_NODE**

enumerator **NVML_TOPOLOGY_SYSTEM**

enum **nvmlIntNvLinkDeviceType_t**

Enum to represent NvLink's remote device type.

Values:

enumerator **NVML_NVLINK_DEVICE_TYPE_GPU**

enumerator **NVML_NVLINK_DEVICE_TYPE_IBMNPU**

enumerator **NVML_NVLINK_DEVICE_TYPE_SWITCH**

enumerator **NVML_NVLINK_DEVICE_TYPE_UNKNOWN**

enum **nvmlNvLinkCapability_t**

Enum to represent NvLink queryable capabilities.

Values:

enumerator **NVML_NVLINK_CAP_P2P_SUPPORTED**

enumerator **NVML_NVLINK_CAP_SYSMEM_ACCESS**

enumerator **NVML_NVLINK_CAP_P2P_ATOMICS**

enumerator **NVML_NVLINK_CAP_SYSMEM_ATOMICS**

enumerator **NVML_NVLINK_CAP_SLI_BRIDGE**

enumerator **NVML_NVLINK_CAP_VALID**

enumerator **NVML_NVLINK_CAP_COUNT**

enum **nvmlNvLinkErrorCounter_t**

Enum to represent NvLink queryable error counters.

Values:

enumerator **NVML_NVLINK_ERROR_DL_REPLAY**

enumerator **NVML_NVLINK_ERROR_DL_RECOVERY**

enumerator **NVML_NVLINK_ERROR_DL_CRC_FLIT**

enumerator **NVML_NVLINK_ERROR_DL_CRC_DATA**

enumerator **NVML_NVLINK_ERROR_DL_ECC_DATA**

enumerator **NVML_NVLINK_ERROR_COUNT**

enum **nvmlNvLinkUtilizationCountPktTypes_t**

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

Values:

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_NOP**

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_READ**

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_WRITE**

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_RATOM**

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_NRATOM**

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_FLUSH**

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_RESPDATA**

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_RESPNODATA**

enumerator **NVML_NVLINK_COUNTER_PKTFILTER_ALL**

enum **nvmlNvLinkUtilizationCountUnits_t**

Enum to represent the NvLink utilization counter packet units.

Values:

enumerator **NVML_NVLINK_COUNTER_UNIT_CYCLES**

enumerator **NVML_NVLINK_COUNTER_UNIT_PACKETS**

enumerator **NVML_NVLINK_COUNTER_UNIT_BYTES**

enumerator **NVML_NVLINK_COUNTER_UNIT_RESERVED**

enumerator **NVML_NVLINK_COUNTER_UNIT_COUNT**

enum **nvmlPcieUtilCounter_t**

Represents the queryable PCIe utilization counters.

Values:

enumerator **NVML_PCIE_UTIL_TX_BYTES**

enumerator **NVML_PCIE_UTIL_RX_BYTES**

enumerator **NVML_PCIE_UTIL_COUNT**

enum **nvmlPerfPolicyType_t**

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

Values:

enumerator **NVML_PERF_POLICY_POWER**

How long did power violations cause the GPU to be below application clocks.

enumerator **NVML_PERF_POLICY_THERMAL**

How long did thermal violations cause the GPU to be below application clocks.

enumerator **NVML_PERF_POLICY_SYNC_BOOST**

How long did sync boost cause the GPU to be below application clocks.

enumerator **NVML_PERF_POLICY_BOARD_LIMIT**

How long did the board limit cause the GPU to be below application clocks.

enumerator **NVML_PERF_POLICY_LOW_UTILIZATION**

How long did low utilization cause the GPU to be below application clocks.

enumerator **NVML_PERF_POLICY_RELIABILITY**

How long did the board reliability limit cause the GPU to be below application clocks.

enumerator **NVML_PERF_POLICY_TOTAL_APP_CLOCKS**

Total time the GPU was held below application clocks by any limiter (0 - 5 above)

enumerator **NVML_PERF_POLICY_TOTAL_BASE_CLOCKS**

Total time the GPU was held below base clocks.

enumerator **NVML_PERF_POLICY_COUNT**

enum **nvmlProcessMode_t**

Enum to represent process mode.

Values:

enumerator **NVML_PROCESS_MODE_COMPUTE**

Processes with a compute context.

enumerator **NVML_PROCESS_MODE_GRAPHICS**

Processes with a graphics context.

enumerator **NVML_PROCESS_MODE_MPS**

Processes with a MPS (Multi-Process Service) compute context.

enumerator **NVML_PROCESS_MODE_ALL**

All processes running on the GPU (compute, graphics, MPS, and other types)

enumerator **NVML_PROCESS_MODE_MAX**
Maximum value for bounds checking.

enum **nvmlSamplingType_t**
Represents Type of Sampling Event.
Values:

enumerator **NVML_TOTAL_POWER_SAMPLES**
To represent total power drawn by GPU.

enumerator **NVML_GPU_UTILIZATION_SAMPLES**
To represent percent of time during which one or more kernels was executing on the GPU.

enumerator **NVML_MEMORY_UTILIZATION_SAMPLES**
To represent percent of time during which global (device) memory was being read or written.

enumerator **NVML_ENC_UTILIZATION_SAMPLES**
To represent percent of time during which NVENC remains busy.

enumerator **NVML_DEC_UTILIZATION_SAMPLES**
To represent percent of time during which NVDEC remains busy.

enumerator **NVML_PROCESSOR_CLK_SAMPLES**
To represent processor clock samples.

enumerator **NVML_MEMORY_CLK_SAMPLES**
To represent memory clock samples.

enumerator **NVML_MODULE_POWER_SAMPLES**
To represent module power samples for total module starting Grace Hopper.

enumerator **NVML_JPG_UTILIZATION_SAMPLES**
To represent percent of time during which NVJPG remains busy.

enumerator **NVML_OFA_UTILIZATION_SAMPLES**
To represent percent of time during which NVOFA remains busy.

enumerator **NVML_SAMPLINGTYPE_COUNT**

enum **nvmlThermalController_t**
Represents the thermal sensor controllers.
Values:

enumerator **NVML_THERMAL_CONTROLLER_NONE**

enumerator **NVML_THERMAL_CONTROLLER_GPU_INTERNAL**

enumerator **NVML_THERMAL_CONTROLLER_ADM1032**

enumerator **NVML_THERMAL_CONTROLLER_ADT7461**

enumerator **NVML_THERMAL_CONTROLLER_MAX6649**

enumerator **NVML_THERMAL_CONTROLLER_MAX1617**

enumerator **NVML_THERMAL_CONTROLLER_LM99**

enumerator **NVML_THERMAL_CONTROLLER_LM89**

enumerator **NVML_THERMAL_CONTROLLER_LM64**

enumerator **NVML_THERMAL_CONTROLLER_G781**

enumerator **NVML_THERMAL_CONTROLLER_ADT7473**

enumerator **NVML_THERMAL_CONTROLLER_SBMAX6649**

enumerator **NVML_THERMAL_CONTROLLER_VBIOSEVT**

enumerator **NVML_THERMAL_CONTROLLER_OS**

enumerator **NVML_THERMAL_CONTROLLER_NVSYSCON_CANOAS**

enumerator **NVML_THERMAL_CONTROLLER_NVSYSCON_E551**

enumerator **NVML_THERMAL_CONTROLLER_MAX6649R**

enumerator **NVML_THERMAL_CONTROLLER_ADT7473S**

enumerator **NVML_THERMAL_CONTROLLER_UNKNOWN**

enum **nvmlThermalTarget_t**

Represents the thermal sensor targets.

Values:

enumerator **NVML_THERMAL_TARGET_NONE**

enumerator **NVML_THERMAL_TARGET_GPU**
GPU core temperature requires NvPhysicalGpuHandle.

enumerator **NVML_THERMAL_TARGET_MEMORY**
GPU memory temperature requires NvPhysicalGpuHandle.

enumerator **NVML_THERMAL_TARGET_POWER_SUPPLY**
GPU power supply temperature requires NvPhysicalGpuHandle.

enumerator **NVML_THERMAL_TARGET_BOARD**
GPU board ambient temperature requires NvPhysicalGpuHandle.

enumerator **NVML_THERMAL_TARGET_VCD_BOARD**
Visual Computing Device Board temperature requires NvVisualComputingDeviceHandle.

enumerator **NVML_THERMAL_TARGET_VCD_INLET**
Visual Computing Device Inlet temperature requires NvVisualComputingDeviceHandle.

enumerator **NVML_THERMAL_TARGET_VCD_OUTLET**
Visual Computing Device Outlet temperature requires NvVisualComputingDeviceHandle.

enumerator **NVML_THERMAL_TARGET_ALL**

enumerator **NVML_THERMAL_TARGET_UNKNOWN**

enum **nvmlUUIDType_t**
Enum to represent different UUID types.
Values:

enumerator **NVML_UUID_TYPE_NONE**
Undefined type.

enumerator **NVML_UUID_TYPE_ASCII**
ASCII format type.

enumerator **NVML_UUID_TYPE_BINARY**
Binary format type.

enum **nvmlValueType_t**
Represents the type for sample value returned.
Values:

enumerator **NVML_VALUE_TYPE_DOUBLE**
Double-precision floating-point value.

enumerator **NVML_VALUE_TYPE_UNSIGNED_INT**
Unsigned 32-bit integer value.

enumerator **NVML_VALUE_TYPE_UNSIGNED_LONG**
Unsigned long integer value.

enumerator **NVML_VALUE_TYPE_UNSIGNED_LONG_LONG**
Unsigned 64-bit integer value.

enumerator **NVML_VALUE_TYPE_SIGNED_LONG_LONG**
Signed 64-bit integer value.

enumerator **NVML_VALUE_TYPE_SIGNED_INT**
Signed 32-bit integer value.

enumerator **NVML_VALUE_TYPE_UNSIGNED_SHORT**
Unsigned 16-bit integer value.

enumerator **NVML_VALUE_TYPE_COUNT**
Maximum value above +1.

5.2.1.8.8 Typedefs

typedef *nvmlCoolerInfo_v1_t* **nvmlCoolerInfo_t**

typedef *nvmlDeviceAddressingMode_v1_t* **nvmlDeviceAddressingMode_t**

typedef struct nvmlDevice_st ***nvmlDevice_t**

typedef struct nvmlGpuInstance_st ***nvmlGpuInstance_t**

typedef *nvmlPciInfoExt_v1_t* **nvmlPciInfoExt_t**

typedef *nvmlPdi_v1_t* **nvmlPdi_t**

typedef *nvmlProcessDetailList_v1_t* **nvmlProcessDetailList_t**

typedef *nvmlRepairStatus_v1_t* **nvmlRepairStatus_t**

typedef *nvmlUUID_v1_t* **nvmlUUID_t**

5.2.1.9 Drain State definitions

5.2.1.9.1 Enumerations

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

nvmlPcieLinkState_t
Parent bridge PCIe link state requested by nvmlDeviceRemoveGpu()

5.2.1.9.2 Enumerations

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

Values:

enumerator **NVML_DETACH_GPU_KEEP**

enumerator **NVML_DETACH_GPU_REMOVE**

enum **nvmlPcieLinkState_t**
Parent bridge PCIe link state requested by nvmlDeviceRemoveGpu()

Values:

enumerator **NVML_PCIE_LINK_KEEP**

enumerator **NVML_PCIE_LINK_SHUT_DOWN**

5.2.1.10 Drain states

This chapter describes methods that NVML can perform against each device to control their drain state and recognition by NVML and NVIDIA kernel driver.

These methods can be used with out-of-band tools to power on/off GPUs, enable robust reset scenarios, etc.

5.2.1.10.1 Functions

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceModifyDrainState*(nvmlPciInfo_t *pciInfo, nvmlEnableState_t newState)
Modify the drain state of a GPU.

nvmlReturn_t *nvmlDeviceQueryDrainState*(nvmlPciInfo_t *pciInfo, nvmlEnableState_t *currentState)
Query the drain state of a GPU.

nvmlReturn_t *nvmlDeviceRemoveGpu_v2*(nvmlPciInfo_t *pciInfo, nvmlDetachGpuState_t gpuState, nvmlPcieLinkState_t linkState)
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.

5.2.1.10.2 Functions

nvmlReturn_t **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 Pascal or newer 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

nvmlReturn_t **nvmlDeviceModifyDrainState**(
nvmlPciInfo_t *pciInfo, *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 no longer see this GPU. Persistence mode for this GPU must be turned off before this call is made. Must be called as administrator. For Linux only.

For Pascal or newer fully supported devices. Some Kepler devices supported.

Parameters

- **pciInfo** – The PCI address of the GPU drain state to be modified
- **newState** – The drain state that should be entered, see *nvmlEnableState_t*

Returns

- *NVML_SUCCESS* if counters were successfully reset
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *nvmlIndex* or *newState* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device doesn't support this feature
- *NVML_ERROR_NO_PERMISSION* if the calling process has insufficient permissions to perform operation
- *NVML_ERROR_IN_USE* if the device has persistence mode turned on
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceQueryDrainState**(
    nvmlPciInfo_t *pciInfo, *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 Pascal or newer fully supported devices. Some Kepler devices supported.

Parameters

- **pciInfo** – The PCI address of the GPU drain state to be queried
- **currentState** – The current drain state for this GPU, see *nvmlEnableState_t*

Returns

- *NVML_SUCCESS* if counters were successfully reset
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *nvmlIndex* or *currentState* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device doesn't support this feature
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceRemoveGpu_v2**(
    nvmlPciInfo_t *pciInfo, *nvmlDetachGpuState_t* gpuState, *nvmlPcieLinkState_t* linkState,
)

This method will remove the specified GPU from the view of both NVML and the NVIDIA kernel driver as long as no other processes are attached.

If other processes are attached, this call will return NVML_ERROR_IN_USE and the GPU will be returned to its original "draining" state. Note: the only situation where a process can still be attached after *nvmlDeviceModifyDrainState()* is called to initiate the draining state is if that process was using, and is still using, a GPU before the call was made. Also note, persistence mode counts as an attachment to the GPU thus it must be disabled prior to this call.

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

For Pascal or newer fully supported devices. Some Kepler devices supported.

Parameters

- **pciInfo** – The PCI address of the GPU to be removed
- **gpuState** – Whether the GPU is to be removed, from the OS see *nvmlDetachGpuState_t*
- **linkState** – Requested upstream PCIe link state, see *nvmlPcieLinkState_t*

Returns

- *NVML_SUCCESS* if counters were successfully reset
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *nvmlIndex* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device doesn't support this feature
- *NVML_ERROR_IN_USE* if the device is still in use and cannot be removed

5.2.1.11 Encoder Structs

5.2.1.11.1 Enumerations

nvmlEncoderType_t

Represents type of encoder for capacity can be queried.

5.2.1.11.2 Structs

nvmlEncoderSessionInfo_t

Structure to hold encoder session data.

5.2.1.11.3 Enumerations

enum **nvmlEncoderType_t**

Represents type of encoder for capacity can be queried.

Values:

enumerator **NVML_ENCODER_QUERY_H264**

H264 encoder.

enumerator **NVML_ENCODER_QUERY_HEVC**

HEVC encoder.

enumerator **NVML_ENCODER_QUERY_AV1**

AV1 encoder.

enumerator **NVML_ENCODER_QUERY_UNKNOWN**

Unknown encoder.

5.2.1.12 Error reporting

This chapter describes helper functions for error reporting routines.

5.2.1.12.1 Functions

const char * *nvmlErrorString*(nvmlReturn_t result)
Helper method for converting NVML error codes into readable strings.

5.2.1.12.2 Functions

const char ***nvmlErrorString**(*nvmlReturn_t* result)
Helper method for converting NVML error codes into readable strings.

For all products.

> **Note**
>
> This helper is available in the MPS locality-domain view described in *MPS Locality Domain Device Enumeration*.

Parameters
result – NVML error code to convert

Returns
String representation of the error.

5.2.1.13 Event Handling Methods

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

5.2.1.13.1 Event Types

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*

5.2.1.13.1.1 Macros

nvmlEventMigConfigChange
Event about MIG configuration changes.

nvmlEventTypeAll
Mask of all events.

nvmlEventTypeClock
Event about clock changes.

nvmlEventTypeDoubleBitEccError
Event about double bit ECC errors.

nvmlEventTypeDramRetirementEvent
Event about DRAM retirement event.

nvmlEventTypeDramRetirementFailure
Event about DRAM retirement failure.

nvmlEventTypeFatalPoisonError
Event for Fatal Poison.

nvmlEventTypeGpuRecoveryAction
Event for GPU Recovery Action.

nvmlEventTypeGpuUnavailableError
Event for GPU Unavailable.

nvmlEventTypeNonFatalPoisonError
Event for Non Fatal Poison.

nvmlEventTypeNone
Mask with no events.

nvmlEventTypePState
Event about PState changes.

nvmlEventTypePowerSourceChange
Event about AC/Battery power source changes.

nvmlEventTypeSingleBitEccError
Event about single bit ECC errors.

nvmlEventTypeSingleBitEccErrorStorm
Event about single bit ECC error storm.

nvmlEventTypeXidCriticalError
Event that Xid critical error occurred.

5.2.1.13.1.2 Macros

nvmlEventMigConfigChange

Event about MIG configuration changes.

MIG configuration change event.

nvmlEventTypeAll

Mask of all events.

Mask of all event types.

nvmlEventTypeClock

Event about clock changes.

Kepler only Clock change event.

nvmlEventTypeDoubleBitEccError

Event about double bit ECC errors.

Note

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

nvmlEventTypeDramRetirementEvent

Event about DRAM retirement event.

DRAM retirement event.

nvmlEventTypeDramRetirementFailure

Event about DRAM retirement failure.

DRAM retirement failure event.

nvmlEventTypeFatalPoisonError

Event for Fatal Poison.

Fatal poison error event.

nvmlEventTypeGpuRecoveryAction

Event for GPU Recovery Action.

GPU recovery action event.

nvmlEventTypeGpuUnavailableError

Event for GPU Unavailable.

GPU unavailable error event.

nvmlEventTypeNonFatalPoisonError

Event for Non Fatal Poison.

Non-fatal poison error event.

nvmlEventTypeNone

Mask with no events.

No event.

nvmlEventTypePState

Event about PState changes.

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. PState change event.

nvmlEventTypePowerSourceChange

Event about AC/Battery power source changes.

Power source change event.

nvmlEventTypeSingleBitEccError

Event about single bit ECC errors.

> **Note**
>
> A corrected texture memory error is not an ECC error, so it does not generate a single bit event Single bit ECC error event.

nvmlEventTypeSingleBitEccErrorStorm

Event about single bit ECC error storm.

Single bit ECC error storm event.

nvmlEventTypeXidCriticalError

Event that Xid critical error occurred.

XID critical error event.

5.2.1.13.2 Macros

NVML_COMPUTE_INSTANCE_ID_ANY
Sentinel value used when no MIG compute instance ID applies.

NVML_GPU_INSTANCE_ID_ANY
Sentinel value used when no MIG GPU instance ID applies.

NVML_OPERATIONAL_EVENT_ATTR_COMPONENT_RESET
Event involved a component reset.

NVML_OPERATIONAL_EVENT_ATTR_LATENT
Event reports a latent condition.

NVML_OPERATIONAL_EVENT_ATTR_OVERFLOW
Source reported that event records were dropped at the source.

NVML_OPERATIONAL_EVENT_ATTR_PRIMARY
Event is the primary event in its group.

NVML_OPERATIONAL_EVENT_ATTR_PROPAGATED
Event was propagated from another source.

NVML_OPERATIONAL_EVENT_ATTR_THRESHOLD_EXCEEDED
Event reports an exceeded threshold.

NVML_OPERATIONAL_EVENT_ATTR_UNCONTAINED
Event reports an uncontained condition.

NVML_OPERATIONAL_EVENT_GROUP_ATTR_PREVERR
Event group reports a previous error condition.

NVML_OPERATIONAL_EVENT_GROUP_ATTR_RECOVERED
Event group reports a recovered condition.

NVML_OPERATIONAL_EVENT_GROUP_ATTR_SIMULATED
Event group was generated by simulation or testing.

nvmlSystemEventSetCreateRequest_v1

nvmlSystemEventSetFreeRequest_v1

nvmlSystemEventSetWaitRequest_v1

nvmlSystemEventTypeCount
Number of system event types.

nvmlSystemEventTypeGpuDriverBind
Bitmask value of Driver Bind System Event.

nvmlSystemEventTypeGpuDriverUnbind
System Event for GPU Driver Unbind.

nvmlSystemRegisterEventRequest_v1

5.2.1.13.3 Enumerations

nvmlEventDataType_t
Event data formats returned by *nvmlEventSetWait_v3* .

nvmlGpuOperationalEventContextType_t
NVML-defined GPU Operational Event context classifications.

nvmlGpuOperationalEventLogLevel_t
Log-level values used by GPU Operational Events.

nvmlOperationalEventSeverity_t
Severity values used by Operational Events.

5.2.1.13.4 Functions

nvmlReturn_t *nvmlDeviceGetSupportedEventTypes*(nvmlDevice_t device, unsigned long long *eventTypes)
Returns information about events supported on device.

nvmlReturn_t *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* .

nvmlReturn_t *nvmlEventSetCreate*(nvmlEventSet_t *set)
Create an empty set of events.

nvmlReturn_t *nvmlEventSetFree*(nvmlEventSet_t set)
Releases events in the set.

nvmlReturn_t *nvmlEventSetGetContextCount_v1*(nvmlEventSet_t set, nvmlEventSetGetContextCount_v1_t *params)
Gets the number of context records for the most recent event returned by *nvmlEventSetWait_v3* on this event set.

nvmlReturn_t *nvmlEventSetGetContextData_v1*(nvmlEventSet_t set, nvmlEventSetGetContextData_v1_t *params)
Copies the raw payload for a context record from the most recent event returned by *nvmlEventSetWait_v3* .

nvmlReturn_t *nvmlEventSetGetContextInfo_v1*(nvmlEventSet_t set, nvmlEventSetGetContextInfo_v1_t *params)
Gets metadata for a context record from the most recent event returned by *nvmlEventSetWait_v3* .

nvmlReturn_t *nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1*(nvmlEventSet_t set, nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t *params)
Gets decoded GPU legacy-Xid context data for a context record from the most recent event returned by *nvmlEventSetWait_v3* .

nvmlReturn_t *nvmlEventSetRegisterGpuOperationalEvents_v1*(nvmlEventSet_t eventSet, const nvmlGpuOperationalEventConfig_v1_t *config)
Adds a GPU Operational Event subscription to an event set.

nvmlReturn_t *nvmlEventSetWait_v2*(nvmlEventSet_t set, nvmlEventData_t *data, unsigned int timeoutms)
Waits on events and delivers events.

nvmlReturn_t *nvmlEventSetWait_v3*(nvmlEventSet_t set, nvmlEventSetWait_v3_t *params)
Waits on an event set and returns the next event in the extended event format.

nvmlReturn_t *nvmlSystemEventSetCreate*(nvmlSystemEventSetCreateRequest_t *request)
Create an empty set of system events.

nvmlReturn_t *nvmlSystemEventSetFree*(nvmlSystemEventSetFreeRequest_t *request)
Releases system event set.

nvmlReturn_t *nvmlSystemEventSetWait*(nvmlSystemEventSetWaitRequest_t *request)
Waits on system events and delivers events.

nvmlReturn_t *nvmlSystemRegisterEvents*(nvmlSystemRegisterEventRequest_t *request)
Starts recording of events on system and add the events to specified *nvmlSystemEventSet_t* .

5.2.1.13.5 Groups

Event Types
Event Types which user can be notified about.

5.2.1.13.6 Structs

nvmlEventData_t
Information about occurred event.

nvmlEventSetGetContextCount_v1_t
Parameters for retrieving the number of context records associated with the most recent event.

nvmlEventSetGetContextData_v1_t
Parameters for retrieving raw context data associated with the most recent event.

nvmlEventSetGetContextInfo_v1_t
Parameters for retrieving context metadata associated with the most recent event.

nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t
Parameters for retrieving decoded GPU legacy-Xid context data.

nvmlSystemEventData_v1_t
nvmlSystemEventData_v1_t

nvmlSystemEventSetCreateRequest_v1_t
nvmlSystemEventSetCreateRequest

nvmlSystemEventSetFreeRequest_v1_t
nvmlSystemEventSetFreeRequest

nvmlSystemEventSetWaitRequest_v1_t
nvmlSystemEventSetWait

nvmlSystemRegisterEventRequest_v1_t
nvmlSystemRegisterEventRequest

5.2.1.13.7 Typedefs

nvmlEventSet_t
Handle to an event set.

nvmlSystemEventSetCreateRequest_t

nvmlSystemEventSetFreeRequest_t

nvmlSystemEventSetWaitRequest_t

nvmlSystemEventSet_t
System Event Set.

nvmlSystemRegisterEventRequest_t

5.2.1.13.8 Macros

NVML_COMPUTE_INSTANCE_ID_ANY
Sentinel value used when no MIG compute instance ID applies.

NVML_GPU_INSTANCE_ID_ANY
Sentinel value used when no MIG GPU instance ID applies.

NVML_OPERATIONAL_EVENT_ATTR_COMPONENT_RESET
Event involved a component reset.

NVML_OPERATIONAL_EVENT_ATTR_LATENT
Event reports a latent condition.

NVML_OPERATIONAL_EVENT_ATTR_OVERFLOW
Source reported that event records were dropped at the source.

NVML_OPERATIONAL_EVENT_ATTR_PRIMARY
Event is the primary event in its group.

NVML_OPERATIONAL_EVENT_ATTR_PROPAGATED
Event was propagated from another source.

NVML_OPERATIONAL_EVENT_ATTR_THRESHOLD_EXCEEDED
Event reports an exceeded threshold.

NVML_OPERATIONAL_EVENT_ATTR_UNCONTAINED
Event reports an uncontained condition.

NVML_OPERATIONAL_EVENT_GROUP_ATTR_PREVERR
Event group reports a previous error condition.

NVML_OPERATIONAL_EVENT_GROUP_ATTR_RECOVERED
Event group reports a recovered condition.

NVML_OPERATIONAL_EVENT_GROUP_ATTR_SIMULATED
Event group was generated by simulation or testing.

nvmlSystemEventSetCreateRequest_v1

nvmlSystemEventSetFreeRequest_v1

nvmlSystemEventSetWaitRequest_v1

nvmlSystemEventTypeCount
Number of system event types.

nvmlSystemEventTypeGpuDriverBind
Bitmask value of Driver Bind System Event.

nvmlSystemEventTypeGpuDriverUnbind
System Event for GPU Driver Unbind.

Bitmask value of Driver Unbind System Event

nvmlSystemRegisterEventRequest_v1

5.2.1.13.9 Enumerations

enum **nvmlEventDataType_t**
Event data formats returned by *nvmlEventSetWait_v3*.

Values:

enumerator **NVML_EVENT_DATA_TYPE_NVML_EVENT**
NVML event-bit data. `eventType` contains an NVML event bit.

enumerator **NVML_EVENT_DATA_TYPE_GPU_OPERATIONAL_EVENT**

Structured GPU Operational Event data.

enum **nvmlGpuOperationalEventContextType_t**

NVML-defined GPU Operational Event context classifications.

These values describe the NVML public interpretation of a context payload. The original source-defined context type is returned separately in *nvmlEventSetGetContextInfo_v1_t::sourceEventContextType*.

Values:

enumerator **NVML_GPU_OPERATIONAL_EVENT_CONTEXT_TYPE_UNKNOWN**

No NVML public interpretation is defined for this context payload.

enumerator **NVML_GPU_OPERATIONAL_EVENT_CONTEXT_TYPE_LEGACY_XID**

Context payload can be decoded with *nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1*.

enum **nvmlGpuOperationalEventLogLevel_t**

Log-level values used by GPU Operational Events.

These values are used both for event reporting in *nvmlEventSetWait_v3_t* and for subscription filtering in *nvmlGpuOperationalEventConfig_v1_t*. Higher numeric values represent more selective log levels. `NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_ALL` disables log-level filtering when used as a subscription threshold. Event data may contain newer log-level values that are not named in this header; clients should handle unrecognized numeric values.

Values:

enumerator **NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_ALL**

Matches all GPU Operational Event log levels.

enumerator **NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_TELEMETRY**

High-volume telemetry events.

enumerator **NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_DIAG**

Diagnostic events.

enumerator **NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_NOTICE**

Notable operational events.

enumerator **NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_WARNING**

Warning events.

enumerator **NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_ERROR**

Error events.

enum **nvmlOperationalEventSeverity_t**

Severity values used by Operational Events.

These values are used both for event reporting in *nvmlEventSetWait_v3_t* and for subscription filtering in *nvmlGpuOperationalEventConfig_v1_t*. Higher numeric values represent more selective severities. NVML_OPERATIONAL_EVENT_SEVERITY_ALL disables severity filtering when used as a subscription threshold. Event data may contain newer severity values that are not named in this header; clients should handle unrecognized numeric values.

Values:

enumerator **NVML_OPERATIONAL_EVENT_SEVERITY_ALL**

Matches all Operational Event severities.

enumerator **NVML_OPERATIONAL_EVENT_SEVERITY_INFORMATIONAL**

Informational event.

enumerator **NVML_OPERATIONAL_EVENT_SEVERITY_CORRECTED**

Corrected error event.

enumerator **NVML_OPERATIONAL_EVENT_SEVERITY_RECOVERABLE**

Recoverable error event.

enumerator **NVML_OPERATIONAL_EVENT_SEVERITY_FATAL**

Fatal error event.

5.2.1.13.10 Functions

nvmlReturn_t **nvmlDeviceGetSupportedEventTypes**(
nvmlDevice_t device, unsigned long long *eventTypes,
)

Returns information about events supported on device.

For Fermi or newer fully supported devices.

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

See also

Event Types

See also

nvmlDeviceRegisterEvents

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

nvmlReturn_t **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 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.

This call starts recording of events on specific device. All events that occurred before this call are not recorded. Checking if some event occurred can be done with *nvmlEventSetWait_v2*

If function reports NVML_ERROR_UNKNOWN, event set is in undefined state and should be freed. If function reports NVML_ERROR_NOT_SUPPORTED, event set can still be used. None of the requested eventTypes are registered in that case.

See also

Event Types

See also

nvmlDeviceGetSupportedEventTypes

See also

nvmlEventSetWait

See also

nvmlEventSetFree

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

nvmlReturn_t **nvmlEventSetCreate**(*nvmlEventSet_t* *set)

Create an empty set of events.

Event set should be freed by *nvmlEventSetFree*

For Fermi or newer fully supported devices.

See also

nvmlEventSetFree

Parameters

set – Reference in which to return the event handle

Returns

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

nvmlReturn_t **nvmlEventSetFree**(*nvmlEventSet_t* set)

Releases events in the set.

For Fermi or newer fully supported devices.

See also

nvmlDeviceRegisterEvents

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

nvmlReturn_t **nvmlEventSetGetContextCount_v1**(
nvmlEventSet_t set, *nvmlEventSetGetContextCount_v1_t* *params,
)

Gets the number of context records for the most recent event returned by *nvmlEventSetWait_v3* on this event set.

This count is tied to the event set, not to a caller-owned copy of *nvmlEventSetWait_v3_t*. It is replaced by the next successful call to *nvmlEventSetWait_v3* on the same event set.

For Turing or newer fully supported devices.

For Linux only.

See also

nvmlEventSetWait_v3

See also

nvmlEventSetGetContextInfo_v1

See also

nvmlEventSetGetContextData_v1

Parameters

- **set** – **[in]** Event set previously used with *nvmlEventSetWait_v3*
- **params** – **[out]** Parameters in which to return the number of context records

Returns

- *NVML_SUCCESS* if *params* has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *set* or *params* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if this API is not available on the platform or driver

- *NVML_ERROR_NOT_FOUND* if no event has been returned by *nvmlEventSetWait_v3*
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlEventSetGetContextData_v1**(
nvmlEventSet_t set, *nvmlEventSetGetContextData_v1_t* *params,
)

Copies the raw payload for a context record from the most recent event returned by *nvmlEventSetWait_v3*.

Passing `params->data` as NULL performs a size query. In that case `params->dataSize` is set to the required payload size and the function returns *NVML_ERROR_INSUFFICIENT_SIZE* when the payload is non-empty. If the context payload is empty, `params->dataSize` is set to 0 and the function returns *NVML_SUCCESS*.

For Turing or newer fully supported devices.

For Linux only.

See also

nvmlEventSetWait_v3

See also

nvmlEventSetGetContextCount_v1

See also

nvmlEventSetGetContextInfo_v1

Parameters

- **set** – **[in]** Event set previously used with *nvmlEventSetWait_v3*
- **params** – **[inout]** Context index, optional data buffer, and buffer size

Returns

- *NVML_SUCCESS* if the raw context payload has been copied
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if arguments are invalid or `params->index` is out of range
- *NVML_ERROR_NOT_FOUND* if no event has been returned by *nvmlEventSetWait_v3*
- *NVML_ERROR_INSUFFICIENT_SIZE* if `params->data` is NULL or too small for a non-empty payload

- *NVML_ERROR_NOT_SUPPORTED* if this API is not available on the platform or driver
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlEventSetGetContextInfo_v1**(
    nvmlEventSet_t set, *nvmlEventSetGetContextInfo_v1_t* *params,
)

Gets metadata for a context record from the most recent event returned by *nvmlEventSetWait_v3*.

The returned metadata identifies the NVML public interpretation, the source-defined context payload type, payload size, and payload format version. The raw payload for any context record can be copied with *nvmlEventSetGetContextData_v1*. When `nvmlGpuOperationalEventContextType` names a type-specific accessor, callers may use that accessor instead. Metadata is replaced by the next successful call to *nvmlEventSetWait_v3* on the same event set.

For Turing or newer fully supported devices.

For Linux only.

See also

nvmlEventSetGetContextInfo_v1_t

See also

nvmlEventSetGetContextCount_v1

See also

nvmlEventSetGetContextData_v1

Parameters

- **`set`** – **[in]** Event set previously used with *nvmlEventSetWait_v3*
- **`params`** – **[inout]** Context index and returned context metadata

Returns

- *NVML_SUCCESS* if the context metadata has been returned
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if arguments are invalid or `params->index` is out of range
- *NVML_ERROR_NOT_FOUND* if no event has been returned by *nvmlEventSetWait_v3*

- *NVML_ERROR_NOT_SUPPORTED* if this API is not available on the platform or driver
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1**(
nvmlEventSet_t set, *nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t* *params,
)

Gets decoded GPU legacy-Xid context data for a context record from the most recent event returned by *nvmlEventSetWait_v3*.

This helper succeeds only for context records whose `nvmlGpuOperationalEventContextType` is *NVML_GPU_OPERATIONAL_EVENT_CONTEXT_TYPE_LEGACY_XID*. Other context records remain available through *nvmlEventSetGetContextData_v1*.

For Turing or newer fully supported devices.

For Linux only.

See also

nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t

See also

nvmlEventSetGetContextInfo_v1

See also

nvmlEventSetGetContextData_v1

Parameters

- **set** – **[in]** Event set previously used with *nvmlEventSetWait_v3*
- **params** – **[inout]** Context index and returned legacy-Xid data

Returns

- *NVML_SUCCESS* if the GPU legacy-Xid context has been returned
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if arguments are invalid or `params->index` is out of range
- *NVML_ERROR_NOT_FOUND* if no event has been returned or the context is not a GPU legacy-Xid context
- *NVML_ERROR_NOT_SUPPORTED* if this API is not available on the platform or driver
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlEventSetRegisterGpuOperationalEvents_v1**(
nvmlEventSet_t eventSet, const *nvmlGpuOperationalEventConfig_v1_t* *config,
)

Adds a GPU Operational Event subscription to an event set.

This API is separate from *nvmlDeviceRegisterEvents*. Calling this API opts the event set into the structured GPU Operational Event format for the target GPU UUID. Subscriptions are identified by *config*; registering the same subscription more than once is treated as success.

nvmlDeviceRegisterEvents and *nvmlEventSetRegisterGpuOperationalEvents_v1* may both be used on the same event set. In that mixed-subscription model, NVML event-bit subscriptions continue to deliver event bits such as *nvmlEventTypeXidCriticalError*, while GPU Operational Event subscriptions deliver *NVML_EVENT_DATA_TYPE_GPU_OPERATIONAL_EVENT* records through *nvmlEventSetWait_v3* with `eventType` set to *nvmlEventTypeNone*. The same underlying incident may generate both an NVML event-bit notification and a structured notification.

This API supports GPU UUID subscriptions.

For Turing or newer fully supported devices.

For Linux only.

See also

nvmlGpuOperationalEventConfig_v1_t

See also

nvmlEventSetWait_v3

See also

nvmlEventSetFree

Parameters

- **`eventSet`** – **[in]** Event set created by *nvmlEventSetCreate*
- **`config`** – **[in]** GPU Operational Event subscription configuration

Returns

- *NVML_SUCCESS* if the GPU Operational Event subscription was registered
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *eventSet* or *config* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if structured GPU Operational Events are not supported on this platform, or if the requested subscription is not supported
- *NVML_ERROR_NO_PERMISSION* if the caller lacks permission for the requested scope

- *NVML_ERROR_INSUFFICIENT_RESOURCES* if the event set cannot accept another subscription
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlEventSetWait_v2**(
nvmlEventSet_t set, *nvmlEventData_t* *data, unsigned int timeoutms,
)

Waits on events and delivers events.

For Fermi or newer fully supported devices.

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

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

On Linux, every Xid error event would return the associated event data and other information if applicable.

In MIG mode, if device handle is provided, the API reports all the events for the available instances, only if the caller has appropriate privileges. In absence of required privileges, only the events which affect all the instances (i.e. whole device) are reported.

This API does not currently support per-instance event reporting using MIG device handles.

See also

Event Types

See also

nvmlDeviceRegisterEvents

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 *set* or *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

nvmlReturn_t **nvmlEventSetWait_v3**(
    nvmlEventSet_t set, *nvmlEventSetWait_v3_t* *params,
)

Waits on an event set and returns the next event in the extended event format.

This API is the unified wait surface for NVML event-bit subscriptions registered with *nvmlDeviceRegisterEvents* and structured GPU Operational Event subscriptions registered with *nvmlEventSetRegisterGpuOperationalEvents_v1*.

The returned format is distinguished by `dataType` in *params*. If `dataType` is *NVML_EVENT_DATA_TYPE_NVML_EVENT*, the event came from the *nvmlDeviceRegisterEvents* path, `eventType` is an NVML event bit such as *nvmlEventTypeXidCriticalError*, and the existing fields preserve their historical semantics. If `dataType` is *NVML_EVENT_DATA_TYPE_GPU_OPERATIONAL_EVENT*, the event came from the structured format, `eventType` is *nvmlEventTypeNone*, and the structured metadata fields are populated.

When an event set contains only NVML event-bit subscriptions, this API normalizes those events into *nvmlEventSetWait_v3_t*. When an event set contains both NVML event-bit and structured subscriptions, each successful call returns the next available event from either path. Clients should branch on `dataType` to determine which format was returned. An event set is not required to have structured GPU Operational Event subscriptions to be used with this API.

Context records for the returned event, if any, are made available through *nvmlEventSetGetContextCount_v1*, *nvmlEventSetGetContextInfo_v1*, and *nvmlEventSetGetContextData_v1*. Context records remain associated with the event set until the next successful call to *nvmlEventSetWait_v3* on the same event set or until the event set is freed.

For Turing or newer fully supported devices.

For Linux only.

See also

nvmlEventSetWait_v3_t

See also

nvmlDeviceRegisterEvents

See also

nvmlEventSetRegisterGpuOperationalEvents_v1

See also

nvmlEventSetGetContextCount_v1

Parameters

- **set** – **[in]** Reference to set of events to wait on
- **params** – **[inout]** Wait parameters and returned event data

Returns

- *NVML_SUCCESS* if the event data has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *set* or *params* is NULL
- *NVML_ERROR_TIMEOUT* if no event arrived in specified timeout or interrupt arrived
- *NVML_ERROR_NOT_SUPPORTED* if this API is not available on the platform or driver
- *NVML_ERROR_MEMORY* if system memory is insufficient
- *NVML_ERROR_GPU_IS_LOST* if a GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlSystemEventSetCreate**(
    nvmlSystemEventSetCreateRequest_t *request,
)

Create an empty set of system events.

Event set should be freed by *nvmlSystemEventSetFree*

For Fermi or newer fully supported devices.

See also

nvmlSystemEventSetFree

Parameters

request – Reference to nvmlSystemEventSetCreateRequest_t

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 request is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* for unsupported version
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlSystemEventSetFree**(
    nvmlSystemEventSetFreeRequest_t *request,
)

Releases system event set.

For Fermi or newer fully supported devices.

See also

nvmlDeviceRegisterEvents

Parameters

request – Reference to nvmlSystemEventSetFreeRequest_t

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 request is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* for unsupported version
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlSystemEventSetWait**(
    nvmlSystemEventSetWaitRequest_t *request,
)

Waits on system events and delivers events.

For Fermi or newer fully supported devices.

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

if the return request->numEvent equals to request->dataSize, there might be outstanding event, it is recommended to call nvmlSystemEventSetWait again to query all the events.

See also

nvmlSystemEventType

See also

nvmlSystemRegisterEvents

Parameters

request – Reference in which to nvmlSystemEventSetWaitRequest_t

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 request is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* for unsupported version
- *NVML_ERROR_TIMEOUT* if no event notification after timeoutms
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlSystemRegisterEvents**(
    nvmlSystemRegisterEventRequest_t *request,
)

Starts recording of events on system and add the events to specified *nvmlSystemEventSet_t*.

For Linux only.

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 *nvmlSystemEventSetWait*

If function reports NVML_ERROR_UNKNOWN, event set is in undefined state and should be freed. If function reports NVML_ERROR_NOT_SUPPORTED, event set can still be used. None of the requested eventTypes are registered in that case.

See also

nvmlSystemEventType

See also

nvmlSystemEventSetWait

See also

nvmlEventSetFree

Parameters

request – Reference to the struct nvmlSystemRegisterEventRequest_t

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 request is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* for unsupported version
- *NVML_ERROR_UNKNOWN* on any unexpected error

5.2.1.13.11 Typedefs

typedef struct nvmlEventSet_st ***nvmlEventSet_t***

Handle to an event set.

typedef *nvmlSystemEventSetCreateRequest_v1_t* **nvmlSystemEventSetCreateRequest_t**

typedef *nvmlSystemEventSetFreeRequest_v1_t* **nvmlSystemEventSetFreeRequest_t**

typedef *nvmlSystemEventSetWaitRequest_v1_t* **nvmlSystemEventSetWaitRequest_t**

typedef struct nvmlSystemEventSet_st ***nvmlSystemEventSet_t***

System Event Set.

typedef *nvmlSystemRegisterEventRequest_v1_t* **nvmlSystemRegisterEventRequest_t**

5.2.1.14 Excluded GPU Queries

This chapter describes NVML operations that are associated with excluded GPUs.

5.2.1.14.1 Functions

nvmlReturn_t *nvmlGetExcludedDeviceCount*(unsigned int *deviceCount)

Retrieves the number of excluded GPU devices in the system.

nvmlReturn_t *nvmlGetExcludedDeviceInfoByIndex*(unsigned int index, nvmlExcludedDeviceInfo_t *info)

Acquire the device information for an excluded GPU device, based on its index.

5.2.1.14.2 Structs

nvmlExcludedDeviceInfo_t

Excluded GPU device information.

5.2.1.14.3 Functions

nvmlReturn_t **nvmlGetExcludedDeviceCount**(unsigned int *deviceCount)

Retrieves the number of excluded GPU devices in the system.

For all products.

Parameters

deviceCount – Reference in which to return the number of excluded devices

Returns

- *NVML_SUCCESS* if *deviceCount* has been set
- *NVML_ERROR_INVALID_ARGUMENT* if *deviceCount* is NULL

nvmlReturn_t **nvmlGetExcludedDeviceInfoByIndex**(
unsigned int index, *nvmlExcludedDeviceInfo_t* *info,
)

Acquire the device information for an excluded GPU device, based on its index.

For all products.

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

See also

nvmlGetExcludedDeviceCount

Parameters

- **index** – The index of the target GPU, >= 0 and < *deviceCount*
- **info** – Reference in which to return the device information

Returns

- *NVML_SUCCESS* if *device* has been set
- *NVML_ERROR_INVALID_ARGUMENT* if *index* is invalid or *info* is NULL

5.2.1.15 Fabric definitions

5.2.1.15.1 Macros

NVML_GPU_FABRIC_CLIQUE_MAX
Maximum number of fabric clique entries.

NVML_GPU_FABRIC_CLIQUE_TYPE_MULTICAST_LOGICAL_ENDPOINT
Multicast logical-endpoint-based access.

NVML_GPU_FABRIC_CLIQUE_TYPE_MULTICAST_POINTER
Multicast pointer-based access.

NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_LOGICAL_ENDPOINT
Unicast logical-endpoint-based access.

NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_POINTER
Unicast pointer-based access.

NVML_GPU_FABRIC_HEALTH_GET
GPU Fabric Health Status Mask for various fields can be obtained using the below macro.

NVML_GPU_FABRIC_HEALTH_MASK_ACCESS_TIMEOUT_RECOVERY_FALSE
Fabric Health Mask: Access Timeout Recovery not in progress.

NVML_GPU_FABRIC_HEALTH_MASK_ACCESS_TIMEOUT_RECOVERY_NOT_SUPPORTED
Fabric Access Timeout Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_ACCESS_TIMEOUT_RECOVERY_TRUE
Fabric Health Mask: Access Timeout Recovery in progress.

NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_FALSE
Fabric Health Mask: Bandwidth not degraded.

NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_NOT_SUPPORTED
Fabric Degraded BW.

NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_TRUE
Fabric Health Mask: Bandwidth degraded.

NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_CONNECTED
Fabric Health Mask: Global Fabric Manager State is Connected.

NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_DISCONNECTED
Fabric Health Mask: Global Fabric Manager State is Disconnected.

NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_NOT_SUPPORTED
Global Fabric Manager State.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_GPU_STATE_INVALID
Fabric Health Mask: Incorrect Configuration - GPU State Invalid.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INCOMPATIBLE_GPU_FW
Fabric Health Mask: Incorrect Configuration - Incompatible GPU Firmware.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INCORRECT_CHASSIS_SN
Fabric Health Mask: Incorrect Configuration - Chassis Serial Number.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INCORRECT_SYSGUID
Fabric Health Mask: Incorrect Configuration - SysGUID.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INSUFFICIENT_NVLINKS
Fabric Health Mask: Incorrect Configuration - Insufficient Nvlinks.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INVALID_LOCATION
Fabric Health Mask: Incorrect Configuration - Invalid Location.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_NONE
Fabric Health Mask: Correct Configuration.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_NOT_SUPPORTED
Fabric Incorrect Configuration.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_NO_PARTITION
Fabric Health Mask: Incorrect Configuration - No Partition.

NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_FALSE
Fabric Health Mask: Partition is not Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_NOT_SUPPORTED
Fabric Partition Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_TRUE
Fabric Health Mask: Partition is Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_FALSE
Fabric Health Mask: Route Recovery not in progress.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_NOT_SUPPORTED
Fabric Route Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_TRUE
Fabric Health Mask: Route Recovery in progress.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_FALSE
Fabric Health Mask: Route is healthy.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_NOT_SUPPORTED
Nvlink Fabric Route Unhealthy.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_TRUE
Fabric Health Mask: Route is unhealthy.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ACCESS_TIMEOUT_RECOVERY
Fabric Health Mask Bit Shift for Access Timeout Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_DEGRADED_BW
Fabric Health Mask Bit Shift for Degraded Bandwidth.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_GFM_STATE
Fabric Health Mask Bit Shift for Global Fabric Manager State.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_INCORRECT_CONFIGURATION
Fabric Health Mask Bit Shift for Incorrect Configuration.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_PARTITION_ASSIGNED
Fabric Health Mask Bit Shift for Partition Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ROUTE_RECOVERY
Fabric Health Mask Bit Shift for Route Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ROUTE_UNHEALTHY
Fabric Health Mask Bit Shift for Route Unhealthy.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ACCESS_TIMEOUT_RECOVERY
Fabric Health Mask Width for Access Timeout Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_DEGRADED_BW
Fabric Health Mask Width for Degraded Bandwidth.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_GFM_STATE
Fabric Health Mask Width for Global Fabric Manager State.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_INCORRECT_CONFIGURATION
Fabric Health Mask Width for Incorrect Configuration.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_PARTITION_ASSIGNED
Fabric Health Mask Width for Partition Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ROUTE_RECOVERY
Fabric Health Mask Width for Route Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ROUTE_UNHEALTHY
Fabric Health Mask Width for Route Unhealthy.

NVML_GPU_FABRIC_HEALTH_SUMMARY_HEALTHY
Fabric Health Summary: Healthy.

NVML_GPU_FABRIC_HEALTH_SUMMARY_LIMITED_CAPACITY
Fabric Health Summary: Limited Capacity.

NVML_GPU_FABRIC_HEALTH_SUMMARY_NOT_SUPPORTED
Fabric Health.

NVML_GPU_FABRIC_HEALTH_SUMMARY_UNHEALTHY
Fabric Health Summary: Unhealthy.

NVML_GPU_FABRIC_HEALTH_TEST
GPU Fabric Health Status Mask for various fields can be tested using the below macro.

NVML_GPU_FABRIC_STATE_COMPLETED
Fabric Probe State completed.

NVML_GPU_FABRIC_STATE_IN_PROGRESS
Fabric Probe in progress.

NVML_GPU_FABRIC_STATE_NOT_STARTED
Fabric Probe has not started.

NVML_GPU_FABRIC_STATE_NOT_SUPPORTED
Fabric Probe States.

NVML_GPU_FABRIC_UUID_LEN
Length of Fabric UUID.

nvmlGpuFabricInfo_v2
Version identifier value for *nvmlGpuFabricInfo_v2_t::version* .

nvmlGpuFabricInfo_v3
Version identifier value for *nvmlGpuFabricInfo_v3_t::version* .

5.2.1.15.2 Structs

nvmlGpuFabricClique_v1_t
Fabric clique entry.

nvmlGpuFabricInfo_t
Contains the device fabric information.

nvmlGpuFabricInfo_v2_t
GPU Fabric information (v2).

nvmlGpuFabricInfo_v3_t
GPU Fabric information (v3).

nvmlGpuFabricInfo_v4_t
GPU Fabric information (v4).

5.2.1.15.3 Typedefs

nvmlGpuFabricInfoV_t

nvmlGpuFabricState_t
Probe State of GPU registration process.

5.2.1.15.4 Macros

NVML_GPU_FABRIC_CLIQUE_MAX
Maximum number of fabric clique entries.

NVML_GPU_FABRIC_CLIQUE_TYPE_MULTICAST_LOGICAL_ENDPOINT
Multicast logical-endpoint-based access.

NVML_GPU_FABRIC_CLIQUE_TYPE_MULTICAST_POINTER
Multicast pointer-based access.

NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_LOGICAL_ENDPOINT

Unicast logical-endpoint-based access.

NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_POINTER

Unicast pointer-based access.

NVML_GPU_FABRIC_HEALTH_GET(var, type)

GPU Fabric Health Status Mask for various fields can be obtained using the below macro.

Ex - *NVML_GPU_FABRIC_HEALTH_GET(var, _DEGRADED_BW)* Macro to get GPU fabric health status.

NVML_GPU_FABRIC_HEALTH_MASK_ACCESS_TIMEOUT_RECOVERY_FALSE

Fabric Health Mask: Access Timeout Recovery not in progress.

NVML_GPU_FABRIC_HEALTH_MASK_ACCESS_TIMEOUT_RECOVERY_NOT_SUPPORTED

Fabric Access Timeout Recovery.

Fabric Health Mask: Access Timeout Recovery not supported

NVML_GPU_FABRIC_HEALTH_MASK_ACCESS_TIMEOUT_RECOVERY_TRUE

Fabric Health Mask: Access Timeout Recovery in progress.

NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_FALSE

Fabric Health Mask: Bandwidth not degraded.

NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_NOT_SUPPORTED

Fabric Degraded BW.

Fabric Health Mask: Degraded Bandwidth not supported

NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_TRUE

Fabric Health Mask: Bandwidth degraded.

NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_CONNECTED

Fabric Health Mask: Global Fabric Manager State is Connected.

NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_DISCONNECTED

Fabric Health Mask: Global Fabric Manager State is Disconnected.

NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_NOT_SUPPORTED

Global Fabric Manager State.

Fabric Health Mask: Global Fabric Manager State not supported

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_GPU_STATE_INVALID

Fabric Health Mask: Incorrect Configuration - GPU State Invalid.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INCOMPATIBLE_GPU_FW
Fabric Health Mask: Incorrect Configuration - Incompatible GPU Firmware.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INCORRECT_CHASSIS_SN
Fabric Health Mask: Incorrect Configuration - Chassis Serial Number.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INCORRECT_SYSGUID
Fabric Health Mask: Incorrect Configuration - SysGUID.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INSUFFICIENT_NVLINKS
Fabric Health Mask: Incorrect Configuration - Insufficient Nvlinks.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_INVALID_LOCATION
Fabric Health Mask: Incorrect Configuration - Invalid Location.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_NONE
Fabric Health Mask: Correct Configuration.

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_NOT_SUPPORTED
Fabric Incorrect Configuration.

Fabric Health Mask: Incorrect Configuration not supported

NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_NO_PARTITION
Fabric Health Mask: Incorrect Configuration - No Partition.

NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_FALSE
Fabric Health Mask: Partition is not Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_NOT_SUPPORTED
Fabric Partition Assigned.

Fabric Health Mask: Partition Assigned not supported

NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_TRUE
Fabric Health Mask: Partition is Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_FALSE
Fabric Health Mask: Route Recovery not in progress.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_NOT_SUPPORTED
Fabric Route Recovery.

Fabric Health Mask: Route Recovery not supported

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_TRUE
Fabric Health Mask: Route Recovery in progress.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_FALSE
Fabric Health Mask: Route is healthy.

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_NOT_SUPPORTED
Nvlink Fabric Route Unhealthy.
Fabric Health Mask: Route Unhealthy not supported

NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_TRUE
Fabric Health Mask: Route is unhealthy.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ACCESS_TIMEOUT_RECOVERY
Fabric Health Mask Bit Shift for Access Timeout Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_DEGRADED_BW
Fabric Health Mask Bit Shift for Degraded Bandwidth.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_GFM_STATE
Fabric Health Mask Bit Shift for Global Fabric Manager State.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_INCORRECT_CONFIGURATION
Fabric Health Mask Bit Shift for Incorrect Configuration.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_PARTITION_ASSIGNED
Fabric Health Mask Bit Shift for Partition Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ROUTE_RECOVERY
Fabric Health Mask Bit Shift for Route Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ROUTE_UNHEALTHY
Fabric Health Mask Bit Shift for Route Unhealthy.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ACCESS_TIMEOUT_RECOVERY
Fabric Health Mask Width for Access Timeout Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_DEGRADED_BW
Fabric Health Mask Width for Degraded Bandwidth.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_GFM_STATE
Fabric Health Mask Width for Global Fabric Manager State.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_INCORRECT_CONFIGURATION
Fabric Health Mask Width for Incorrect Configuration.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_PARTITION_ASSIGNED
Fabric Health Mask Width for Partition Assigned.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ROUTE_RECOVERY

Fabric Health Mask Width for Route Recovery.

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ROUTE_UNHEALTHY

Fabric Health Mask Width for Route Unhealthy.

NVML_GPU_FABRIC_HEALTH_SUMMARY_HEALTHY

Fabric Health Summary: Healthy.

NVML_GPU_FABRIC_HEALTH_SUMMARY_LIMITED_CAPACITY

Fabric Health Summary: Limited Capacity.

NVML_GPU_FABRIC_HEALTH_SUMMARY_NOT_SUPPORTED

Fabric Health.

Fabric Health Summary: Not supported

NVML_GPU_FABRIC_HEALTH_SUMMARY_UNHEALTHY

Fabric Health Summary: Unhealthy.

NVML_GPU_FABRIC_HEALTH_TEST(var, type, val)

GPU Fabric Health Status Mask for various fields can be tested using the below macro.

Ex - *NVML_GPU_FABRIC_HEALTH_TEST(var, _DEGRADED_BW, _TRUE)* Macro to test GPU fabric health status.

NVML_GPU_FABRIC_STATE_COMPLETED

Fabric Probe State completed.

NVML_GPU_FABRIC_STATE_IN_PROGRESS

Fabric Probe in progress.

NVML_GPU_FABRIC_STATE_NOT_STARTED

Fabric Probe has not started.

NVML_GPU_FABRIC_STATE_NOT_SUPPORTED

Fabric Probe States.

Fabric Probe State not supported

NVML_GPU_FABRIC_UUID_LEN

Length of Fabric UUID.

nvmlGpuFabricInfo_v2

Version identifier value for *nvmlGpuFabricInfo_v2_t::version*.

Version macro for nvmlGpuFabricInfo_v2_t

nvmlGpuFabricInfo_v3

Version identifier value for *nvmlGpuFabricInfo_v3_t::version*.

Version macro for nvmlGpuFabricInfo_v3_t

5.2.1.15.5 Typedefs

typedef *nvmlGpuFabricInfo_v3_t* **nvmlGpuFabricInfoV_t**

typedef unsigned char **nvmlGpuFabricState_t**

Probe State of GPU registration process.

5.2.1.16 Field Value Enums

5.2.1.16.1 Macros

NVML_C2C_POWER_STATE_FULL_POWER

NVML_C2C_POWER_STATE_LOW_POWER

NVML_FI_DEV_ACTIVE_BANK_REMAPPINGS
Number of active bank remappings.

NVML_FI_DEV_BANK_REMAPPER_HISTOGRAM_MAX
Number of groups with full bank remap availability.

NVML_FI_DEV_BANK_REMAPPER_HISTOGRAM_NONE
Number of groups with no spare bankremap availability.

NVML_FI_DEV_C2C_LINK_COUNT
Number of C2C Links present on the device.

NVML_FI_DEV_C2C_LINK_ERROR_INTR
C2C Link CRC Error Counter.

NVML_FI_DEV_C2C_LINK_ERROR_REPLAY
C2C Link Replay Error Counter.

NVML_FI_DEV_C2C_LINK_ERROR_REPLAY_B2B
C2C Link Back to Back Replay Error Counter.

NVML_FI_DEV_C2C_LINK_GET_MAX_BW
C2C Link Speed in MBps for active links.

NVML_FI_DEV_C2C_LINK_GET_STATUS
C2C Link Status 0=INACTIVE 1=ACTIVE.

NVML_FI_DEV_C2C_LINK_POWER_STATE
C2C Link Power state. See NVML_C2C_POWER_STATE_*.

NVML_FI_DEV_CLOCKS_EVENT_REASON_HW_POWER_BRAKE_SLOWDOWN
Throttling due to external power brake assertion trigger (reducing core clocks by a factor of 2 or more) in ns.

NVML_FI_DEV_CLOCKS_EVENT_REASON_HW_THERM_SLOWDOWN
Throttling due to temperature being too high (reducing core clocks by a factor of 2 or more) in ns.

NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_POWER_CAP
Field values for Clock Throttle Reason Counters All counters are in nanoseconds.

NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_THERM_SLOWDOWN
Throttling to ensure ((GPU temp < GPU Max Operating Temp) && (Memory Temp < Memory Max Operating Temp)) in ns.

NVML_FI_DEV_CLOCKS_EVENT_REASON_SYNC_BOOST
Throttling to match minimum possible clock across Sync Boost Group in ns.

NVML_FI_DEV_DRAIN_AND_RESET_STATUS
Deprecated, do not use (use NVML_FI_DEV_GET_GPU_RECOVERY_ACTION instead)

NVML_FI_DEV_ECC_CURRENT
Field Identifiers.

NVML_FI_DEV_ECC_DBE_AGG_CBU
CBU double bit aggregate ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_DEV
Device memory double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_L1
L1 cache double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_L2
L2 cache double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_REG
Register File double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_TEX
Texture memory double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_TOTAL
Total double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_CBU
CBU double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_DEV
Device memory double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_L1
L1 cache double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_L2
L2 cache double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_REG
Register file double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_TEX
Texture memory double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_TOTAL
Total double bit volatile ECC errors.

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

NVML_FI_DEV_ECC_SBE_AGG_DEV
Device memory single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_L1
L1 cache single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_L2
L2 cache single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_REG
Register File single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_TEX
Texture memory single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_TOTAL
Total single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_DEV
Device memory single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_L1
L1 cache single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_L2
L2 cache single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_REG
Register file single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_TEX
Texture memory single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_TOTAL
Total single bit volatile ECC errors.

NVML_FI_DEV_EDPP_MULTIPLIER
EDPp multiplier expressed as a percentage.

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

NVML_FI_DEV_GET_GPU_RECOVERY_ACTION
GPU Recovery action. See *nvmlDeviceGpuRecoveryAction_t* .

NVML_FI_DEV_GPU_TEMP
Current GPU temperature in degrees C with sub-degree precision.

NVML_FI_DEV_INACTIVE_BANK_REMAPPINGS
Number of inactive bank remappings.

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

NVML_FI_DEV_IS_RESETLESS_MIG_SUPPORTED
Device's Restless MIG Capability.

NVML_FI_DEV_MCLK_MIN_SWITCH_INTERVAL_MILLISECONDS
minimum required elapsed time between runtime mclk switches, 0 = no rate limit

NVML_FI_DEV_MCLK_SWITCH_TYPE
See NVML_MCLK_SWITCH_TYPE_<XYZ> for all enumerations.

NVML_FI_DEV_MEMORY_TEMP
Memory temperature for the device.

NVML_FI_DEV_MMA_STALL_PERCENT
MMA (Matrix Multiply Accumulate) stall percentage.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L0
NVLink Bandwidth Counter for Counter Set 0, Lane 0.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L1
NVLink Bandwidth Counter for Counter Set 0, Lane 1.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L10
NVLink Bandwidth Counter for Counter Set 0, Lane 10.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L11
NVLink Bandwidth Counter for Counter Set 0, Lane 11.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L2
NVLink Bandwidth Counter for Counter Set 0, Lane 2.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L3
NVLink Bandwidth Counter for Counter Set 0, Lane 3.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L4
NVLink Bandwidth Counter for Counter Set 0, Lane 4.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L5
NVLink Bandwidth Counter for Counter Set 0, Lane 5.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L6
NVLink Bandwidth Counter for Counter Set 0, Lane 6.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L7
NVLink Bandwidth Counter for Counter Set 0, Lane 7.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L8
NVLink Bandwidth Counter for Counter Set 0, Lane 8.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L9
NVLink Bandwidth Counter for Counter Set 0, Lane 9.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_TOTAL
NVLink Bandwidth Counter Total for Counter Set 0, All Lanes.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L0
NVLink Bandwidth Counter for Counter Set 1, Lane 0.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L1
NVLink Bandwidth Counter for Counter Set 1, Lane 1.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L10
NVLink Bandwidth Counter for Counter Set 1, Lane 10.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L11
NVLink Bandwidth Counter for Counter Set 1, Lane 11.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L2
NVLink Bandwidth Counter for Counter Set 1, Lane 2.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L3
NVLink Bandwidth Counter for Counter Set 1, Lane 3.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L4
NVLink Bandwidth Counter for Counter Set 1, Lane 4.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L5
NVLink Bandwidth Counter for Counter Set 1, Lane 5.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L6
NVLink Bandwidth Counter for Counter Set 1, Lane 6.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L7
NVLink Bandwidth Counter for Counter Set 1, Lane 7.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L8
NVLink Bandwidth Counter for Counter Set 1, Lane 8.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L9
NVLink Bandwidth Counter for Counter Set 1, Lane 9.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_TOTAL
NVLink Bandwidth Counter Total for Counter Set 1, All Lanes.

NVML_FI_DEV_NVLINK_COUNT_BUFFER_OVERRUN_ERRORS
Number of packets that were discarded on Rx due to buffer overrun.

NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_BER
NVLink Effective BER.

NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_ERRORS
Sum of the number of errors in each Nvlink packet.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_0
NVLink counter field id 235-250.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_1
Count of symbol errors that are corrected - bin 1.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_10
Count of symbol errors that are corrected - bin 10.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_11
Count of symbol errors that are corrected - bin 11.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_12
Count of symbol errors that are corrected - bin 12.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_13
Count of symbol errors that are corrected - bin 13.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_14
Count of symbol errors that are corrected - bin 14.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_15
Count of symbol errors that are corrected - bin 15.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_2
Count of symbol errors that are corrected - bin 2.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_3
Count of symbol errors that are corrected - bin 3.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_4
Count of symbol errors that are corrected - bin 4.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_5
Count of symbol errors that are corrected - bin 5.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_6
Count of symbol errors that are corrected - bin 6.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_7
Count of symbol errors that are corrected - bin 7.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_8
Count of symbol errors that are corrected - bin 8.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_9
Count of symbol errors that are corrected - bin 9.

NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_EVENTS
Number of times link went from Up to recovery, irrespective of the result.

NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_FAILED_EVENTS
Number of times link went from Up to recovery, failed and link was declared down.

NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_SUCCESSFUL_EVENTS
Number of times link went from Up to recovery, succeeded and link came back up.

NVML_FI_DEV_NVLINK_COUNT_LOCAL_LINK_INTEGRITY_ERRORS
Total number of times that the count of local errors exceeded a threshold.

NVML_FI_DEV_NVLINK_COUNT_MALFORMED_PACKET_ERRORS
Number of packets Rx on a link where packets are malformed.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER
Deprecated, do not use.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE0
Deprecated, do not use.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE0_V2
NVLINK raw BER for lane 0.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE1
Deprecated, do not use.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE1_V2
NVLINK raw BER for lane 1.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_V2
NVLINK total raw BER.

NVML_FI_DEV_NVLINK_COUNT_RAW_ERRORS_LANE0
NVLINK raw error count for lane 0.

NVML_FI_DEV_NVLINK_COUNT_RAW_ERRORS_LANE1
NVLINK raw error count for lane 1.

NVML_FI_DEV_NVLINK_COUNT_RCV_BYTES
Total Rx bytes on the link in NVLink5.

NVML_FI_DEV_NVLINK_COUNT_RCV_ERRORS
Total number of packets with errors Rx on a link.

NVML_FI_DEV_NVLINK_COUNT_RCV_GENERAL_ERRORS
Total number of packets Rx with header mismatch.

NVML_FI_DEV_NVLINK_COUNT_RCV_PACKETS
Total Rx packets on the link in NVLink5.

NVML_FI_DEV_NVLINK_COUNT_RCV_REMOTE_ERRORS
Total number of packets Rx - stomp/EBP marker.

NVML_FI_DEV_NVLINK_COUNT_SYMBOL_BER
NVLink Symbol BER.

NVML_FI_DEV_NVLINK_COUNT_SYMBOL_ERRORS
Number of errors in rx symbols.

NVML_FI_DEV_NVLINK_COUNT_VL15_DROPPED
Deprecated, do not use.

NVML_FI_DEV_NVLINK_COUNT_XMIT_BYTES
Total Tx bytes on the link in NVLink5.

NVML_FI_DEV_NVLINK_COUNT_XMIT_DISCARDS
Total number of tx error packets that were discarded.

NVML_FI_DEV_NVLINK_COUNT_XMIT_PACKETS
NVLink counter field id 201-225.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L0
NVLink CRC Data Error Counters.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L1
NVLink data CRC Error Counter for Lane 1.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L10
NVLink data CRC Error Counter for Lane 10.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L11
NVLink data CRC Error Counter for Lane 11.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L2
NVLink data CRC Error Counter for Lane 2.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L3
NVLink data CRC Error Counter for Lane 3.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L4
NVLink data CRC Error Counter for Lane 4.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L5
NVLink data CRC Error Counter for Lane 5.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L6
NVLink CRC Data Error Counters.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L7
NVLink data CRC Error Counter for Lane 7.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L8
NVLink data CRC Error Counter for Lane 8.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L9
NVLink data CRC Error Counter for Lane 9.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_TOTAL
NvLink data CRC Error Counter total for all Lanes.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L0
NVLink Flit Error Counters.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L1
NVLink flow control CRC Error Counter for Lane 1.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L10
NVLink flow control CRC Error Counter for Lane 10.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L11
NVLink flow control CRC Error Counter for Lane 11.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L2
NVLink flow control CRC Error Counter for Lane 2.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L3
NVLink flow control CRC Error Counter for Lane 3.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L4
NVLink flow control CRC Error Counter for Lane 4.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L5
NVLink flow control CRC Error Counter for Lane 5.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L6
NVLink Flit Error Counters.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L7
NVLink flow control CRC Error Counter for Lane 7.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L8
NVLink flow control CRC Error Counter for Lane 8.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L9
NVLink flow control CRC Error Counter for Lane 9.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_TOTAL
NVLink flow control CRC Error Counter total for all Lanes.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L0
NVLink data ECC Error Counter for Link 0.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L1
NVLink data ECC Error Counter for Link 1.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L10
NVLink data ECC Error Counter for Link 10.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L11
NVLink data ECC Error Counter for Link 11.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L2
NVLink data ECC Error Counter for Link 2.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L3
NVLink data ECC Error Counter for Link 3.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L4
NVLink data ECC Error Counter for Link 4.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L5
NVLink data ECC Error Counter for Link 5.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L6
NVLink data ECC Error Counter for Link 6.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L7
NVLink data ECC Error Counter for Link 7.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L8
NVLink data ECC Error Counter for Link 8.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L9
NVLink data ECC Error Counter for Link 9.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_TOTAL
NVLink data ECC Error Counter total for all Links.

NVML_FI_DEV_NVLINK_ERROR_DL_CRC
NVLink Recovery Error CRC Counter.

NVML_FI_DEV_NVLINK_ERROR_DL_RECOVERY
NVLink Recovery Error Counter.

NVML_FI_DEV_NVLINK_ERROR_DL_REPLAY
NVLink Error Replay.

NVML_FI_DEV_NVLINK_GET_DATA_RATE
The Effective Nvlink Data rate available for transactions after accounting for FEC overhead.

NVML_FI_DEV_NVLINK_GET_POWER_STATE
NVLink Power state. 0=HIGH_SPEED 1=LOW_SPEED.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD
NVLink length of idle period (units can be found from NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_UNITS) before transitioning links to sleep state.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_MAX
Max Nvlink Power Threshold. See NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_MIN
Min Nvlink Power Threshold. See NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_SUPPORTED
Determine if Nvlink Power Threshold feature is supported.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_UNITS
Values are in the form NVML_NVLINK_LOW_POWER_THRESHOLD_UNIT_*.

NVML_FI_DEV_NVLINK_GET_SPEED
NVLink Speed, State and Version field id 164, 165, and 166.

NVML_FI_DEV_NVLINK_GET_STATE
NVLink State - one of NVML_NVLINK_STATE_* values.

NVML_FI_DEV_NVLINK_GET_VERSION
NVLink Version.

NVML_FI_DEV_NVLINK_LINK_COUNT
Number of NVLinks present on the device.

NVML_FI_DEV_NVLINK_PLR_XMIT_BLOCKS
NVLINK PLR Xmit Blocks.

NVML_FI_DEV_NVLINK_PLR_XMIT_RETRY_BLOCKS
NVLINK PLR Xmit Retry Blocks.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L0
NVLink Recovery Error Counters.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L1
NVLink Recovery Error Counter for Lane 1.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L10
NVLink Recovery Error Counter for Lane 10.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L11
NVLink Recovery Error Counter for Lane 11.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L2
NVLink Recovery Error Counter for Lane 2.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L3
NVLink Recovery Error Counter for Lane 3.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L4
NVLink Recovery Error Counter for Lane 4.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L5
NVLink Recovery Error Counter for Lane 5.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L6
NVLink Recovery Error Counters.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L7
NVLink Recovery Error Counter for Lane 7.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L8
NVLink Recovery Error Counter for Lane 8.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L9
NVLink Recovery Error Counter for Lane 9.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_TOTAL
NVLink Recovery Error Counter total for all Lanes.

NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID
Remote device NVLink ID.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L0
NVLink Replay Error Counters.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L1
NVLink Replay Error Counter for Lane 1.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L10
NVLink Replay Error Counter for Lane 10.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L11
NVLink Replay Error Counter for Lane 11.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L2
NVLink Replay Error Counter for Lane 2.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L3
NVLink Replay Error Counter for Lane 3.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L4
NVLink Replay Error Counter for Lane 4.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L5
NVLink Replay Error Counter for Lane 5.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L6
NVLink Replay Error Counters.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L7
NVLink Replay Error Counter for Lane 7.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L8
NVLink Replay Error Counter for Lane 8.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L9
NVLink Replay Error Counter for Lane 9.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_TOTAL
NVLink Replay Error Counter total for all Lanes.

NVML_FI_DEV_NVLINK_SPEED_MBPS_COMMON
Common NVLink Speed in MBps for active links.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L0
NVLink Speed.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L1
NVLink Speed in MBps for Link 1.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L10
NVLink Speed in MBps for Link 10.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L11
NVLink Speed in MBps for Link 11.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L2
NVLink Speed in MBps for Link 2.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L3
NVLink Speed in MBps for Link 3.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L4
NVLink Speed in MBps for Link 4.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L5
NVLink Speed in MBps for Link 5.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L6
NVLink Speed.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L7
NVLink Speed in MBps for Link 7.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L8
NVLink Speed in MBps for Link 8.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L9
NVLink Speed in MBps for Link 9.

NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_RX
NVLink RX Data throughput in KiB.

NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_TX
NVLink throughput counters field values.

NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_RX
NVLink RX Data + protocol overhead in KiB.

NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_TX
NVLink TX Data + protocol overhead in KiB.

NVML_FI_DEV_NVSWITCH_CONNECTED_LINK_COUNT
NVSwitch: connected NVLink count.

NVML_FI_DEV_PCIE_COUNT_BAD_DLLP
PCIe Bad DLLP Counter.

NVML_FI_DEV_PCIE_COUNT_BAD_TLP
PCIe Bad TLP Counter.

NVML_FI_DEV_PCIE_COUNT_CORRECTABLE_ERRORS
PCIe Correctable Errors Counter.

NVML_FI_DEV_PCIE_COUNT_FATAL_ERROR
PCIe Fatal Error Counter.

NVML_FI_DEV_PCIE_COUNT_LANE_ERROR
PCIe Per Lane Error Counter.

NVML_FI_DEV_PCIE_COUNT_LCRC_ERROR
PCIe LCRC Error Counter.

NVML_FI_DEV_PCIE_COUNT_NAKS_RECEIVED
PCIe NAK Receive Counter.

NVML_FI_DEV_PCIE_COUNT_NAKS_SENT
PCIe NAK Send Counter.

NVML_FI_DEV_PCIE_COUNT_NON_FATAL_ERROR
PCIe Non Fatal Error Counter.

NVML_FI_DEV_PCIE_COUNT_RECEIVER_ERROR
PCIe Receiver Error Counter.

NVML_FI_DEV_PCIE_COUNT_RX_BYTES
PCIe receive bytes. Value can be wrapped.

NVML_FI_DEV_PCIE_COUNT_TX_BYTES
PCIe transmit bytes. Value can be wrapped.

NVML_FI_DEV_PCIE_COUNT_UNSUPPORTED_REQ
PCIe Unsupported Request Counter.

NVML_FI_DEV_PCIE_INBOUND_ATOMICS_MASK

NVML_FI_DEV_PCIE_L0_TO_RECOVERY_COUNTER
Device PEX error recovery counter.

NVML_FI_DEV_PCIE_OUTBOUND_ATOMICS_MASK

NVML_FI_DEV_PCIE_REPLAY_COUNTER
PCIe replay counter.

NVML_FI_DEV_PCIE_REPLAY_ROLLOVER_COUNTER
PCIe replay rollover counter.

NVML_FI_DEV_PENDING_BANK_REMAPPING
If any banks are pending remapping. 1=yes 0=no.

NVML_FI_DEV_PERF_POLICY_BOARD_LIMIT
Perf Policy Counter for Board Limit.

NVML_FI_DEV_PERF_POLICY_LOW_UTILIZATION
Perf Policy Counter for Low GPU Utilization Policy.

NVML_FI_DEV_PERF_POLICY_POWER
Perf Policy Counter for Power Policy.

NVML_FI_DEV_PERF_POLICY_RELIABILITY
Perf Policy Counter for Reliability Policy.

NVML_FI_DEV_PERF_POLICY_SYNC_BOOST
Perf Policy Counter for Sync boost Policy.

NVML_FI_DEV_PERF_POLICY_THERMAL
Perf Policy Counter for Thermal Policy.

NVML_FI_DEV_PERF_POLICY_TOTAL_APP_CLOCKS
Perf Policy Counter for Total App Clock Policy.

NVML_FI_DEV_PERF_POLICY_TOTAL_BASE_CLOCKS
Perf Policy Counter for Total Base Clocks Policy.

NVML_FI_DEV_POWER_AVERAGE
Retrieves power usage for this GPU in milliwatts.

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

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

NVML_FI_DEV_POWER_INSTANT
Current GPU power, supported on all architectures.

NVML_FI_DEV_POWER_MAX_LIMIT
Maximum power limit in milliwatts.

NVML_FI_DEV_POWER_MIN_LIMIT
Minimum power limit in milliwatts.

NVML_FI_DEV_POWER_REQUESTED_LIMIT
Power limit requested by NVML or any other userspace client.

NVML_FI_DEV_POWER_SYNC_BALANCING_AF
Accumulated activity factor of the GPU to be used for averaging.

NVML_FI_DEV_POWER_SYNC_BALANCING_FREQ
Accumulated frequency of the GPU to be used for averaging.

NVML_FI_DEV_REMAPPED_COR
Number of remapped rows due to correctable errors.

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

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

NVML_FI_DEV_REMAPPED_ROWS_COR_INACTIVE
Number of inactive row remappings due to correctable errors.

NVML_FI_DEV_REMAPPED_ROWS_UNC_INACTIVE
Number of inactive row remappings due to uncorrectable errors.

NVML_FI_DEV_REMAPPED_UNC
Number of remapped rows due to uncorrectable errors.

NVML_FI_DEV_RESET_STATUS
Depracated, do not use (use NVML_FI_DEV_GET_GPU_RECOVERY_ACTION instead)

NVML_FI_DEV_RETIRED_DBE
Number of retired pages because of double bit errors.

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

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

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

NVML_FI_DEV_RETIRED_SBE
Number of retired pages because of single bit errors.

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

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

NVML_FI_DEV_TEMPERATURE_SHUTDOWN_TLIMIT
GPU T.Limit temperature thresholds in degree Celsius.

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

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

NVML_FI_MAX
One greater than the largest field ID defined above.

NVML_FI_PWR_SMOOTHING_ACTIVE_PRESET_PROFILE
Active preset profile number.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PERCENT_TMP_FLOOR
% TMP floor for a given profile

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_ACT_OFFSET
Current primary floor activation offset value in Watts for admin override.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_ACT_WIN_MULT
Current primary floor activation window multiplier value for admin override.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_TAR_WIN_MULT
Current primary floor target window multiplier value for admin override.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_DOWN_HYST_VAL
Ramp down hysteresis value in ms for a given profile.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_DOWN_RATE
Ramp down rate in mW/s for a given profile.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_UP_RATE
Ramp up rate in mW/s for a given profile.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_SECONDARY_POWER_FLOOR
Current secondary power floor value in Watts for admin override.

NVML_FI_PWR_SMOOTHING_APPLIED_TMP_CEIL
Applied TMP ceiling value in Watts.

NVML_FI_PWR_SMOOTHING_APPLIED_TMP_FLOOR
Applied TMP floor value in Watts.

NVML_FI_PWR_SMOOTHING_DELAYED_PWR_SMOOTHING_SUPPORTED
Support (0/Not Supported or 1/Supported) for delayed power smoothing.

NVML_FI_PWR_SMOOTHING_ENABLED
Enablement (0/DISABLED or 1/ENABLED)

NVML_FI_PWR_SMOOTHING_HW_CIRCUITRY_PERCENT_LIFETIME_REMAINING
HW Circuitry % lifetime remaining.

NVML_FI_PWR_SMOOTHING_IMM_RAMP_DOWN_ENABLED
Immediate ramp down enablement (0/DISABLED or 1/ENABLED)

NVML_FI_PWR_SMOOTHING_LIMITED_TIME
Running total of time the GPU has been limited by power smoothing in ns.

NVML_FI_PWR_SMOOTHING_MAX_NUM_PRESET_PROFILES
Max number of preset profiles.

NVML_FI_PWR_SMOOTHING_MAX_PERCENT_TMP_FLOOR_SETTING
Max % TMP Floor value.

NVML_FI_PWR_SMOOTHING_MIN_PERCENT_TMP_FLOOR_SETTING
Min % TMP Floor value.

NVML_FI_PWR_SMOOTHING_MIN_PRIMARY_FLOOR_ACT_OFFSET
Minimum primary floor activation offset value in Watts.

NVML_FI_PWR_SMOOTHING_MIN_PRIMARY_FLOOR_ACT_POINT
Minimum primary floor activation point value in Watts.

NVML_FI_PWR_SMOOTHING_PRIMARY_POWER_FLOOR
Current primary power floor value in Watts.

NVML_FI_PWR_SMOOTHING_PRIV_LVL
Current privilege level.

NVML_FI_PWR_SMOOTHING_PROFILE_PERCENT_TMP_FLOOR
% TMP floor for a given profile

NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_ACT_OFFSET
Current primary floor activation offset value in Watts for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_ACT_WIN_MULT
Current primary floor activation window multiplier value for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_TAR_WIN_MULT
Current primary floor target window multiplier value for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_DOWN_HYST_VAL
Ramp down hysteresis value in ms for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_DOWN_RATE
Ramp down rate in mW/s for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_UP_RATE
Ramp up rate in mW/s for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_SECONDARY_POWER_FLOOR
Current secondary power floor value in Watts for a given profile.

NVML_FI_PWR_SMOOTHING_SECONDARY_POWER_FLOOR
Current secondary power floor value in Watts.

NVML_FI_PWR_SMOOTHING_SOC_POWER_SMOOTHING_ENABLED
State-Of-Charge Power Smoothing Enabled (0/DISABLED or 1/ENABLED)

NVML_FI_PWR_SMOOTHING_WINDOW_MULTIPLIER
Window Multiplier value in ms.

NVML_MCLK_SWITCH_TYPE_DEFERRED
deferred switching (driver reload)

NVML_MCLK_SWITCH_TYPE_NOT_SUPPORTED
NVML_FI_DEV_MCLK_SWITCH_TYPE enumerations.

NVML_MCLK_SWITCH_TYPE_RUNTIME
runtime switching

NVML_NVLINK_LOW_POWER_THRESHOLD_DEFAULT

NVML_NVLINK_LOW_POWER_THRESHOLD_MAX

NVML_NVLINK_LOW_POWER_THRESHOLD_MIN

NVML_NVLINK_LOW_POWER_THRESHOLD_RESET

NVML_NVLINK_LOW_POWER_THRESHOLD_UNIT_100US
NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_UNITS.

NVML_NVLINK_LOW_POWER_THRESHOLD_UNIT_50US

NVML_NVLINK_POWER_STATE_HIGH_SPEED
NVML_NVLINK_POWER_STATES.

NVML_NVLINK_POWER_STATE_LOW

5.2.1.16.2 Structs

nvmlFieldValue_t
Information for a Field Value Sample.

nvmlNvLinkPowerThres_t

5.2.1.16.3 Macros

NVML_C2C_POWER_STATE_FULL_POWER

NVML_C2C_POWER_STATE_LOW_POWER

NVML_FI_DEV_ACTIVE_BANK_REMAPPINGS
Number of active bank remappings.

NVML_FI_DEV_BANK_REMAPPER_HISTOGRAM_MAX
Number of groups with full bank remap availability.

NVML_FI_DEV_BANK_REMAPPER_HISTOGRAM_NONE
Number of groups with no spare bankremap availability.

NVML_FI_DEV_C2C_LINK_COUNT
Number of C2C Links present on the device.

NVML_FI_DEV_C2C_LINK_ERROR_INTR
C2C Link CRC Error Counter.

NVML_FI_DEV_C2C_LINK_ERROR_REPLAY
C2C Link Replay Error Counter.

NVML_FI_DEV_C2C_LINK_ERROR_REPLAY_B2B
C2C Link Back to Back Replay Error Counter.

NVML_FI_DEV_C2C_LINK_GET_MAX_BW
C2C Link Speed in MBps for active links.

NVML_FI_DEV_C2C_LINK_GET_STATUS
C2C Link Status 0=INACTIVE 1=ACTIVE.

NVML_FI_DEV_C2C_LINK_POWER_STATE
C2C Link Power state. See NVML_C2C_POWER_STATE_*.

NVML_FI_DEV_CLOCKS_EVENT_REASON_HW_POWER_BRAKE_SLOWDOWN
Throttling due to external power brake assertion trigger (reducing core clocks by a factor of 2 or more) in ns.

NVML_FI_DEV_CLOCKS_EVENT_REASON_HW_THERM_SLOWDOWN
Throttling due to temperature being too high (reducing core clocks by a factor of 2 or more) in ns.

NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_POWER_CAP

Field values for Clock Throttle Reason Counters All counters are in nanoseconds.

Throttling to not exceed currently set power limits in ns

NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_THERM_SLOWDOWN

Throttling to ensure ((GPU temp < GPU Max Operating Temp) && (Memory Temp < Memory Max Operating Temp)) in ns.

NVML_FI_DEV_CLOCKS_EVENT_REASON_SYNC_BOOST

Throttling to match minimum possible clock across Sync Boost Group in ns.

NVML_FI_DEV_DRAIN_AND_RESET_STATUS

Deprecated, do not use (use NVML_FI_DEV_GET_GPU_RECOVERY_ACTION instead)

NVML_FI_DEV_ECC_CURRENT

Field Identifiers.

All Identifiers pertain to a device. Each ID is only used once and is guaranteed never to change. Current ECC mode. 1=Active. 0=Inactive

NVML_FI_DEV_ECC_DBE_AGG_CBU

CBU double bit aggregate ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_DEV

Device memory double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_L1

L1 cache double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_L2

L2 cache double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_REG

Register File double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_TEX

Texture memory double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_AGG_TOTAL

Total double bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_CBU

CBU double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_DEV

Device memory double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_L1
L1 cache double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_L2
L2 cache double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_REG
Register file double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_TEX
Texture memory double bit volatile ECC errors.

NVML_FI_DEV_ECC_DBE_VOL_TOTAL
Total double bit volatile ECC errors.

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

NVML_FI_DEV_ECC_SBE_AGG_DEV
Device memory single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_L1
L1 cache single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_L2
L2 cache single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_REG
Register File single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_TEX
Texture memory single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_AGG_TOTAL
Total single bit aggregate (persistent) ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_DEV
Device memory single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_L1
L1 cache single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_L2
L2 cache single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_REG
Register file single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_TEX
Texture memory single bit volatile ECC errors.

NVML_FI_DEV_ECC_SBE_VOL_TOTAL
Total single bit volatile ECC errors.

NVML_FI_DEV_EDPP_MULTIPLIER
EDPp multiplier expressed as a percentage.

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

NVML_FI_DEV_GET_GPU_RECOVERY_ACTION
GPU Recovery action. See *nvmlDeviceGpuRecoveryAction_t*.

NVML_FI_DEV_GPU_TEMP
Current GPU temperature in degrees C with sub-degree precision.

NVML_FI_DEV_INACTIVE_BANK_REMAPPINGS
Number of inactive bank remappings.

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

NVML_FI_DEV_IS_RESETLESS_MIG_SUPPORTED
Device's Restless MIG Capability.

NVML_FI_DEV_MCLK_MIN_SWITCH_INTERVAL_MILLISECONDS
minimum required elapsed time between runtime mclk switches, 0 = no rate limit

NVML_FI_DEV_MCLK_SWITCH_TYPE
See NVML_MCLK_SWITCH_TYPE_<XYZ> for all enumerations.

NVML_FI_DEV_MEMORY_TEMP
Memory temperature for the device.

NVML_FI_DEV_MMA_STALL_PERCENT
MMA (Matrix Multiply Accumulate) stall percentage.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L0
NVLink Bandwidth Counter for Counter Set 0, Lane 0.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L1
NVLink Bandwidth Counter for Counter Set 0, Lane 1.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L10
NVLink Bandwidth Counter for Counter Set 0, Lane 10.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L11
NVLink Bandwidth Counter for Counter Set 0, Lane 11.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L2
NVLink Bandwidth Counter for Counter Set 0, Lane 2.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L3
NVLink Bandwidth Counter for Counter Set 0, Lane 3.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L4
NVLink Bandwidth Counter for Counter Set 0, Lane 4.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L5
NVLink Bandwidth Counter for Counter Set 0, Lane 5.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L6
NVLink Bandwidth Counter for Counter Set 0, Lane 6.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L7
NVLink Bandwidth Counter for Counter Set 0, Lane 7.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L8
NVLink Bandwidth Counter for Counter Set 0, Lane 8.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L9
NVLink Bandwidth Counter for Counter Set 0, Lane 9.

NVML_FI_DEV_NVLINK_BANDWIDTH_C0_TOTAL
NVLink Bandwidth Counter Total for Counter Set 0, All Lanes.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L0
NVLink Bandwidth Counter for Counter Set 1, Lane 0.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L1
NVLink Bandwidth Counter for Counter Set 1, Lane 1.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L10
NVLink Bandwidth Counter for Counter Set 1, Lane 10.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L11
NVLink Bandwidth Counter for Counter Set 1, Lane 11.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L2
NVLink Bandwidth Counter for Counter Set 1, Lane 2.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L3
NVLink Bandwidth Counter for Counter Set 1, Lane 3.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L4
NVLink Bandwidth Counter for Counter Set 1, Lane 4.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L5
NVLink Bandwidth Counter for Counter Set 1, Lane 5.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L6
NVLink Bandwidth Counter for Counter Set 1, Lane 6.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L7
NVLink Bandwidth Counter for Counter Set 1, Lane 7.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L8
NVLink Bandwidth Counter for Counter Set 1, Lane 8.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L9
NVLink Bandwidth Counter for Counter Set 1, Lane 9.

NVML_FI_DEV_NVLINK_BANDWIDTH_C1_TOTAL
NVLink Bandwidth Counter Total for Counter Set 1, All Lanes.

NVML_FI_DEV_NVLINK_COUNT_BUFFER_OVERRUN_ERRORS
Number of packets that were discarded on Rx due to buffer overrun.

NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_BER
NVLink Effective BER.

Bit [0:7]: BER Exponent value Bit [8:11]: BER MANTISSA value Use macro NVML_NVLINK_ERROR_COUNTER_BER_GET to extract the two fields Effective BER for effective errors

NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_ERRORS
Sum of the number of errors in each Nvlink packet.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_0
NVLink counter field id 235-250.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. Count of symbol errors that are corrected - bin 0

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_1
Count of symbol errors that are corrected - bin 1.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_10
Count of symbol errors that are corrected - bin 10.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_11
Count of symbol errors that are corrected - bin 11.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_12
Count of symbol errors that are corrected - bin 12.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_13
Count of symbol errors that are corrected - bin 13.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_14
Count of symbol errors that are corrected - bin 14.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_15
Count of symbol errors that are corrected - bin 15.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_2
Count of symbol errors that are corrected - bin 2.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_3
Count of symbol errors that are corrected - bin 3.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_4
Count of symbol errors that are corrected - bin 4.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_5
Count of symbol errors that are corrected - bin 5.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_6
Count of symbol errors that are corrected - bin 6.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_7
Count of symbol errors that are corrected - bin 7.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_8
Count of symbol errors that are corrected - bin 8.

NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_9
Count of symbol errors that are corrected - bin 9.

NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_EVENTS
Number of times link went from Up to recovery, irrespective of the result.

NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_FAILED_EVENTS
Number of times link went from Up to recovery, failed and link was declared down.

NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_SUCCESSFUL_EVENTS
Number of times link went from Up to recovery, succeeded and link came back up.

NVML_FI_DEV_NVLINK_COUNT_LOCAL_LINK_INTEGRITY_ERRORS
Total number of times that the count of local errors exceeded a threshold.

NVML_FI_DEV_NVLINK_COUNT_MALFORMED_PACKET_ERRORS
Number of packets Rx on a link where packets are malformed.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER
Deprecated, do not use.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE0
Deprecated, do not use.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE0_V2
NVLINK raw BER for lane 0.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE1
Deprecated, do not use.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE1_V2
NVLINK raw BER for lane 1.

NVML_FI_DEV_NVLINK_COUNT_RAW_BER_V2
NVLINK total raw BER.

NVML_FI_DEV_NVLINK_COUNT_RAW_ERRORS_LANE0
NVLINK raw error count for lane 0.

NVML_FI_DEV_NVLINK_COUNT_RAW_ERRORS_LANE1
NVLINK raw error count for lane 1.

NVML_FI_DEV_NVLINK_COUNT_RCV_BYTES
Total Rx bytes on the link in NVLink5.

NVML_FI_DEV_NVLINK_COUNT_RCV_ERRORS
Total number of packets with errors Rx on a link.

NVML_FI_DEV_NVLINK_COUNT_RCV_GENERAL_ERRORS

Total number of packets Rx with header mismatch.

NVML_FI_DEV_NVLINK_COUNT_RCV_PACKETS

Total Rx packets on the link in NVLink5.

NVML_FI_DEV_NVLINK_COUNT_RCV_REMOTE_ERRORS

Total number of packets Rx - stomp/EBP marker.

NVML_FI_DEV_NVLINK_COUNT_SYMBOL_BER

NVLink Symbol BER.

Bit [0:7]: BER Exponent value Bit [8:11]: BER MANTISSA value Use macro NVML_NVLINK_ERROR_COUNTER_BER_GET to extract the two fields BER for symbol errors

NVML_FI_DEV_NVLINK_COUNT_SYMBOL_ERRORS

Number of errors in rx symbols.

NVML_FI_DEV_NVLINK_COUNT_VL15_DROPPED

Deprecated, do not use.

NVML_FI_DEV_NVLINK_COUNT_XMIT_BYTES

Total Tx bytes on the link in NVLink5.

NVML_FI_DEV_NVLINK_COUNT_XMIT_DISCARDS

Total number of tx error packets that were discarded.

NVML_FI_DEV_NVLINK_COUNT_XMIT_PACKETS

NVLink counter field id 201-225.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. Total Tx packets on the link in NVLink5

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L0

NVLink CRC Data Error Counters.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink data CRC Error Counter for Lane 0

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L1

NVLink data CRC Error Counter for Lane 1.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L10

NVLink data CRC Error Counter for Lane 10.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L11

NVLink data CRC Error Counter for Lane 11.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L2

NVLink data CRC Error Counter for Lane 2.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L3

NVLink data CRC Error Counter for Lane 3.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L4

NVLink data CRC Error Counter for Lane 4.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L5

NVLink data CRC Error Counter for Lane 5.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L6

NVLink CRC Data Error Counters.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink data CRC Error Counter for Lane 6

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L7

NVLink data CRC Error Counter for Lane 7.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L8

NVLink data CRC Error Counter for Lane 8.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L9

NVLink data CRC Error Counter for Lane 9.

NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_TOTAL

NvLink data CRC Error Counter total for all Lanes.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L0

NVLink Flit Error Counters.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink flow control CRC Error Counter for Lane 0

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L1

NVLink flow control CRC Error Counter for Lane 1.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L10

NVLink flow control CRC Error Counter for Lane 10.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L11

NVLink flow control CRC Error Counter for Lane 11.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L2

NVLink flow control CRC Error Counter for Lane 2.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L3
NVLink flow control CRC Error Counter for Lane 3.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L4
NVLink flow control CRC Error Counter for Lane 4.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L5
NVLink flow control CRC Error Counter for Lane 5.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L6
NVLink Flit Error Counters.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink flow control CRC Error Counter for Lane 6

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L7
NVLink flow control CRC Error Counter for Lane 7.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L8
NVLink flow control CRC Error Counter for Lane 8.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L9
NVLink flow control CRC Error Counter for Lane 9.

NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_TOTAL
NVLink flow control CRC Error Counter total for all Lanes.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L0
NVLink data ECC Error Counter for Link 0.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L1
NVLink data ECC Error Counter for Link 1.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L10
NVLink data ECC Error Counter for Link 10.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L11
NVLink data ECC Error Counter for Link 11.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L2
NVLink data ECC Error Counter for Link 2.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L3
NVLink data ECC Error Counter for Link 3.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L4

NVLink data ECC Error Counter for Link 4.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L5

NVLink data ECC Error Counter for Link 5.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L6

NVLink data ECC Error Counter for Link 6.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L7

NVLink data ECC Error Counter for Link 7.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L8

NVLink data ECC Error Counter for Link 8.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L9

NVLink data ECC Error Counter for Link 9.

NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_TOTAL

NVLink data ECC Error Counter total for all Links.

NVML_FI_DEV_NVLINK_ERROR_DL_CRC

NVLink Recovery Error CRC Counter.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink CRC Error Counter This is unsupported for Blackwell+ Please use NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_*

NVML_FI_DEV_NVLINK_ERROR_DL_RECOVERY

NVLink Recovery Error Counter.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Recovery Error Counter This is unsupported for Blackwell+ Please use NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_*

NVML_FI_DEV_NVLINK_ERROR_DL_REPLAY

NVLink Error Replay.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Replay Error Counter This is unsupported for Blackwell+. Please use NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_*

NVML_FI_DEV_NVLINK_GET_DATA_RATE

The Effective Nvlink Data rate available for transactions after accounting for FEC overhead.

NVML_FI_DEV_NVLINK_GET_POWER_STATE

NVLink Power state. 0=HIGH_SPEED 1=LOW_SPEED.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD

NVLink length of idle period (units can be found from NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_UNITS) before transitioning links to sleep state.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_MAX

Max Nvlink Power Threshold. See NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_MIN

Min Nvlink Power Threshold. See NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_SUPPORTED

Determine if Nvlink Power Threshold feature is supported.

NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_UNITS

Values are in the form NVML_NVLINK_LOW_POWER_THRESHOLD_UNIT_*.

NVML_FI_DEV_NVLINK_GET_SPEED

NVLink Speed, State and Version field id 164, 165, and 166.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Speed in MBps

NVML_FI_DEV_NVLINK_GET_STATE

NVLink State - one of NVML_NVLINK_STATE_* values.

NVML_FI_DEV_NVLINK_GET_VERSION

NVLink Version.

NVML_FI_DEV_NVLINK_LINK_COUNT

Number of NVLinks present on the device.

NVML_FI_DEV_NVLINK_PLR_XMIT_BLOCKS

NVLINK PLR Xmit Blocks.

NVML_FI_DEV_NVLINK_PLR_XMIT_RETRY_BLOCKS

NVLINK PLR Xmit Retry Blocks.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L0

NVLink Recovery Error Counters.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Recovery Error Counter for Lane 0

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L1

NVLink Recovery Error Counter for Lane 1.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L10

NVLink Recovery Error Counter for Lane 10.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L11

NVLink Recovery Error Counter for Lane 11.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L2

NVLink Recovery Error Counter for Lane 2.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L3

NVLink Recovery Error Counter for Lane 3.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L4

NVLink Recovery Error Counter for Lane 4.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L5

NVLink Recovery Error Counter for Lane 5.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L6

NVLink Recovery Error Counters.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Recovery Error Counter for Lane 6

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L7

NVLink Recovery Error Counter for Lane 7.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L8

NVLink Recovery Error Counter for Lane 8.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L9

NVLink Recovery Error Counter for Lane 9.

NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_TOTAL

NVLink Recovery Error Counter total for all Lanes.

NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID

Remote device NVLink ID.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. Remote device NVLink ID

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L0

NVLink Replay Error Counters.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Replay Error Counter for Lane 0

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L1

NVLink Replay Error Counter for Lane 1.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L10

NVLink Replay Error Counter for Lane 10.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L11

NVLink Replay Error Counter for Lane 11.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L2

NVLink Replay Error Counter for Lane 2.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L3

NVLink Replay Error Counter for Lane 3.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L4

NVLink Replay Error Counter for Lane 4.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L5

NVLink Replay Error Counter for Lane 5.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L6

NVLink Replay Error Counters.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Replay Error Counter for Lane 6

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L7

NVLink Replay Error Counter for Lane 7.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L8

NVLink Replay Error Counter for Lane 8.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L9

NVLink Replay Error Counter for Lane 9.

NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_TOTAL

NVLink Replay Error Counter total for all Lanes.

NVML_FI_DEV_NVLINK_SPEED_MBPS_COMMON

Common NVLink Speed in MBps for active links.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L0

NVLink Speed.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Speed in MBps for Link 0

NVML_FI_DEV_NVLINK_SPEED_MBPS_L1

NVLink Speed in MBps for Link 1.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L10

NVLink Speed in MBps for Link 10.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L11

NVLink Speed in MBps for Link 11.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L2

NVLink Speed in MBps for Link 2.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L3

NVLink Speed in MBps for Link 3.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L4

NVLink Speed in MBps for Link 4.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L5

NVLink Speed in MBps for Link 5.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L6

NVLink Speed.

Link ID needs to be specified in the scopeId field in *nvmlFieldValue_t*. NVLink Speed in MBps for Link 6

NVML_FI_DEV_NVLINK_SPEED_MBPS_L7

NVLink Speed in MBps for Link 7.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L8

NVLink Speed in MBps for Link 8.

NVML_FI_DEV_NVLINK_SPEED_MBPS_L9

NVLink Speed in MBps for Link 9.

NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_RX

NVLink RX Data throughput in KiB.

NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_TX

NVLink throughput counters field values.

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

NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_RX

NVLink RX Data + protocol overhead in KiB.

NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_TX
NVLink TX Data + protocol overhead in KiB.

NVML_FI_DEV_NVSWITCH_CONNECTED_LINK_COUNT
NVSwitch: connected NVLink count.

Number of NVLinks connected to NVSwitch

NVML_FI_DEV_PCIE_COUNT_BAD_DLLP
PCIe Bad DLLP Counter.

NVML_FI_DEV_PCIE_COUNT_BAD_TLP
PCIe Bad TLP Counter.

NVML_FI_DEV_PCIE_COUNT_CORRECTABLE_ERRORS
PCIe Correctable Errors Counter.

NVML_FI_DEV_PCIE_COUNT_FATAL_ERROR
PCIe Fatal Error Counter.

NVML_FI_DEV_PCIE_COUNT_LANE_ERROR
PCIe Per Lane Error Counter.

NVML_FI_DEV_PCIE_COUNT_LCRC_ERROR
PCIe LCRC Error Counter.

NVML_FI_DEV_PCIE_COUNT_NAKS_RECEIVED
PCIe NAK Receive Counter.

NVML_FI_DEV_PCIE_COUNT_NAKS_SENT
PCIe NAK Send Counter.

NVML_FI_DEV_PCIE_COUNT_NON_FATAL_ERROR
PCIe Non Fatal Error Counter.

NVML_FI_DEV_PCIE_COUNT_RECEIVER_ERROR
PCIe Receiver Error Counter.

NVML_FI_DEV_PCIE_COUNT_RX_BYTES
PCIe receive bytes. Value can be wrapped.

NVML_FI_DEV_PCIE_COUNT_TX_BYTES
PCIe transmit bytes. Value can be wrapped.

NVML_FI_DEV_PCIE_COUNT_UNSUPPORTED_REQ
PCIe Unsupported Request Counter.

NVML_FI_DEV_PCIE_INBOUND_ATOMICS_MASK

NVML_FI_DEV_PCIE_L0_TO_RECOVERY_COUNTER

Device PEX error recovery counter.

NVML_FI_DEV_PCIE_OUTBOUND_ATOMICS_MASK

NVML_FI_DEV_PCIE_REPLAY_COUNTER

PCIe replay counter.

NVML_FI_DEV_PCIE_REPLAY_ROLLOVER_COUNTER

PCIe replay rollover counter.

NVML_FI_DEV_PENDING_BANK_REMAPPING

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

NVML_FI_DEV_PERF_POLICY_BOARD_LIMIT

Perf Policy Counter for Board Limit.

NVML_FI_DEV_PERF_POLICY_LOW_UTILIZATION

Perf Policy Counter for Low GPU Utilization Policy.

NVML_FI_DEV_PERF_POLICY_POWER

Perf Policy Counter for Power Policy.

NVML_FI_DEV_PERF_POLICY_RELIABILITY

Perf Policy Counter for Reliability Policy.

NVML_FI_DEV_PERF_POLICY_SYNC_BOOST

Perf Policy Counter for Sync boost Policy.

NVML_FI_DEV_PERF_POLICY_THERMAL

Perf Policy Counter for Thermal Policy.

NVML_FI_DEV_PERF_POLICY_TOTAL_APP_CLOCKS

Perf Policy Counter for Total App Clock Policy.

NVML_FI_DEV_PERF_POLICY_TOTAL_BASE_CLOCKS

Perf Policy Counter for Total Base Clocks Policy.

NVML_FI_DEV_POWER_AVERAGE

Retrieves power usage for this GPU in milliwatts.

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

scopeId needs to be specified. It signifies: 0 - GPU Only Scope - Metrics for GPU are retrieved 1 - Module scope - Metrics for the module (e.g. CPU + GPU) are retrieved. 2 - GPU base scope - GPU base Power metrics are retrieved. Note: Only supports _MIN_LIMIT, _MAX_LIMIT, _DEFAULT_LIMIT, and _CURRENT_LIMIT fields. Note: CPU here refers to NVIDIA CPU (e.g. Grace). x86 or non-NVIDIA ARM is not supported GPU power averaged over 1 sec interval, supported on Ampere (except GA100) or newer architectures.

NVML_FI_DEV_POWER_CURRENT_LIMIT

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

NVML_FI_DEV_POWER_DEFAULT_LIMIT

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

NVML_FI_DEV_POWER_INSTANT

Current GPU power, supported on all architectures.

NVML_FI_DEV_POWER_MAX_LIMIT

Maximum power limit in milliwatts.

NVML_FI_DEV_POWER_MIN_LIMIT

Minimum power limit in milliwatts.

NVML_FI_DEV_POWER_REQUESTED_LIMIT

Power limit requested by NVML or any other userspace client.

NVML_FI_DEV_POWER_SYNC_BALANCING_AF

Accumulated activity factor of the GPU to be used for averaging.

NVML_FI_DEV_POWER_SYNC_BALANCING_FREQ

Accumulated frequency of the GPU to be used for averaging.

NVML_FI_DEV_REMAPPED_COR

Number of remapped rows due to correctable errors.

NVML_FI_DEV_REMAPPED_FAILURE

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

NVML_FI_DEV_REMAPPED_PENDING

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

NVML_FI_DEV_REMAPPED_ROWS_COR_INACTIVE

Number of inactive row remappings due to correctable errors.

NVML_FI_DEV_REMAPPED_ROWS_UNC_INACTIVE

Number of inactive row remappings due to uncorrectable errors.

NVML_FI_DEV_REMAPPED_UNC
Number of remapped rows due to uncorrectable errors.

NVML_FI_DEV_RESET_STATUS
Depracated, do not use (use NVML_FI_DEV_GET_GPU_RECOVERY_ACTION instead)

NVML_FI_DEV_RETIRED_DBE
Number of retired pages because of double bit errors.

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

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

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

NVML_FI_DEV_RETIRED_SBE
Number of retired pages because of single bit errors.

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

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

NVML_FI_DEV_TEMPERATURE_SHUTDOWN_TLIMIT
GPU T.Limit temperature thresholds in degree Celsius.

These fields are supported on Ada and later architectures and supersedes *nvmlDeviceGetTemperatureThreshold*. T.Limit temperature after which GPU may shut down for HW protection

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

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

NVML_FI_MAX
One greater than the largest field ID defined above.

NVML_FI_PWR_SMOOTHING_ACTIVE_PRESET_PROFILE
Active preset profile number.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PERCENT_TMP_FLOOR

% TMP floor for a given profile

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_ACT_OFFSET

Current primary floor activation offset value in Watts for admin override.

If the target moving average falls below the secondary floor plus this offset, the primary floor will be activated.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_ACT_WIN_MULT

Current primary floor activation window multiplier value for admin override.

This is the "X" ms time multiplier for the activation moving average window size. The activation moving average is compared against the (secondary floor + primary floor activation offset value) to determine if the controller should switch from the secondary floor to the primary floor. Setting this to 0 will disable switching to the secondary floor.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR_TAR_WIN_MULT

Current primary floor target window multiplier value for admin override.

This is the "X" ms time multiplier for the target moving average window size. When set to non-zero value, the target moving average power determines the primary floor. When set to 0, driver will use the Floor percentage instead to derive the primary floor.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_DOWN_HYST_VAL

Ramp down hysteresis value in ms for a given profile.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_DOWN_RATE

Ramp down rate in mW/s for a given profile.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_UP_RATE

Ramp up rate in mW/s for a given profile.

NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_SECONDARY_POWER_FLOOR

Current secondary power floor value in Watts for admin override.

This is the power floor that will be applied during active workload periods on the GPU when primary floor activation window multiplier is set to a non-zero value.

NVML_FI_PWR_SMOOTHING_APPLIED_TMP_CEIL

Applied TMP ceiling value in Watts.

NVML_FI_PWR_SMOOTHING_APPLIED_TMP_FLOOR

Applied TMP floor value in Watts.

NVML_FI_PWR_SMOOTHING_DELAYED_PWR_SMOOTHING_SUPPORTED

Support (0/Not Supported or 1/Supported) for delayed power smoothing.

NVML_FI_PWR_SMOOTHING_ENABLED

Enablement (0/DISABLED or 1/ENABLED)

NVML_FI_PWR_SMOOTHING_HW_CIRCUITRY_PERCENT_LIFETIME_REMAINING

HW Circuitry % lifetime remaining.

NVML_FI_PWR_SMOOTHING_IMM_RAMP_DOWN_ENABLED

Immediate ramp down enablement (0/DISABLED or 1/ENABLED)

NVML_FI_PWR_SMOOTHING_LIMITED_TIME

Running total of time the GPU has been limited by power smoothing in ns.

NVML_FI_PWR_SMOOTHING_MAX_NUM_PRESET_PROFILES

Max number of preset profiles.

NVML_FI_PWR_SMOOTHING_MAX_PERCENT_TMP_FLOOR_SETTING

Max % TMP Floor value.

NVML_FI_PWR_SMOOTHING_MIN_PERCENT_TMP_FLOOR_SETTING

Min % TMP Floor value.

NVML_FI_PWR_SMOOTHING_MIN_PRIMARY_FLOOR_ACT_OFFSET

Minimum primary floor activation offset value in Watts.

This is the minimum primary floor activation offset accepted by the driver specified in Watts. This is a static field.

NVML_FI_PWR_SMOOTHING_MIN_PRIMARY_FLOOR_ACT_POINT

Minimum primary floor activation point value in Watts.

This is the minimum absolute raw value specified in Watts that the driver will use for switching between primary and secondary floor. This point is calculated as “secondary power floor +

primary floor activation offset”, and then computed value is floored to “min primary floor

activation point” by the driver at run time. This value is used to avoid setting of switch point too low accidentally.

NVML_FI_PWR_SMOOTHING_PRIMARY_POWER_FLOOR

Current primary power floor value in Watts.

This value is calculated by doing “TMP ceiling value * (% TMP floor value)”.

NVML_FI_PWR_SMOOTHING_PRIV_LVL

Current privilege level.

NVML_FI_PWR_SMOOTHING_PROFILE_PERCENT_TMP_FLOOR

% TMP floor for a given profile

NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_ACT_OFFSET

Current primary floor activation offset value in Watts for a given profile.

If the target moving average falls below the secondary floor plus this offset, the primary floor will be activated.

NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_ACT_WIN_MULT

Current primary floor activation window multiplier value for a given profile.

This is the "X" ms time multiplier for the activation moving average window size. The activation moving average is compared against the (secondary floor + primary floor activation offset value) to determine if the controller should switch from the secondary floor to the primary floor. Setting this to 0 will disable switching to the secondary floor.

NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_TAR_WIN_MULT

Current primary floor target window multiplier value for a given profile.

This is the "X" ms time multiplier for the target moving average window size. When set to non-zero value, the target moving average power determines the primary floor. When set to 0, driver will use the Floor percentage instead to derive the primary floor.

NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_DOWN_HYST_VAL

Ramp down hysteresis value in ms for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_DOWN_RATE

Ramp down rate in mW/s for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_UP_RATE

Ramp up rate in mW/s for a given profile.

NVML_FI_PWR_SMOOTHING_PROFILE_SECONDARY_POWER_FLOOR

Current secondary power floor value in Watts for a given profile.

This is the power floor that will be applied during active workload periods on the GPU when primary floor activation window multiplier is set to a non-zero value.

NVML_FI_PWR_SMOOTHING_SECONDARY_POWER_FLOOR

Current secondary power floor value in Watts.

This is the power floor that is applied during active workload periods on the GPU when primary floor activation window multiplier is set to a non-zero value.

NVML_FI_PWR_SMOOTHING_SOC_POWER_SMOOTHING_ENABLED

State-Of-Charge Power Smoothing Enabled (0/DISABLED or 1/ENABLED)

NVML_FI_PWR_SMOOTHING_WINDOW_MULTIPLIER

Window Multiplier value in ms.

This is the multiplier unit specified in ms for other multipliers in the profile (primary floor activation window multiplier and primary floor target window multiplier). This is a static field.

NVML_MCLK_SWITCH_TYPE_DEFERRED
deferred switching (driver reload)

NVML_MCLK_SWITCH_TYPE_NOT_SUPPORTED
NVML_FI_DEV_MCLK_SWITCH_TYPE enumerations.

switching is not supported

NVML_MCLK_SWITCH_TYPE_RUNTIME
runtime switching

NVML_NVLINK_LOW_POWER_THRESHOLD_DEFAULT

NVML_NVLINK_LOW_POWER_THRESHOLD_MAX

NVML_NVLINK_LOW_POWER_THRESHOLD_MIN

NVML_NVLINK_LOW_POWER_THRESHOLD_RESET

NVML_NVLINK_LOW_POWER_THRESHOLD_UNIT_100US
NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_UNITS.

NVML_NVLINK_LOW_POWER_THRESHOLD_UNIT_50US

NVML_NVLINK_POWER_STATE_HIGH_SPEED
NVML_NVLINK_POWER_STATES.

NVML_NVLINK_POWER_STATE_LOW

5.2.1.17 Field Value Queries

This chapter describes NVML operations that are associated with retrieving Field Values from NVML.

5.2.1.17.1 Functions

nvmlReturn_t *nvmlDeviceClearFieldValues*(nvmlDevice_t device, int valuesCount, nvmlFieldValue_t *values)
Clear values for a list of fields for a device.

nvmlReturn_t *nvmlDeviceGetFieldValues*(nvmlDevice_t device, int valuesCount, nvmlFieldValue_t *values)
Request values for a list of fields for a device.

5.2.1.17.2 Functions

nvmlReturn_t **nvmlDeviceClearFieldValues**(
nvmlDevice_t device, int valuesCount, *nvmlFieldValue_t* *values,
)

Clear values for a list of fields for a device.

This API allows multiple fields to be cleared at once.

Parameters

- **device** – The device handle of the GPU to request field values for
- **valuesCount** – Number of entries in values that should be cleared
- **values** – Array of *valuesCount* structures to hold field values. Each value's fieldId must be populated prior to this call

Returns

- *NVML_SUCCESS* if any values in *values* were cleared. Note that you must check the nvmlReturn field of each value for each individual status
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *values* is NULL

nvmlReturn_t **nvmlDeviceGetFieldValues**(
    nvmlDevice_t device, int valuesCount, *nvmlFieldValue_t* *values,
)

Request values for a list of fields for a device.

This API allows multiple fields to be queried at once. If any of the underlying fieldIds are populated by the same driver call, the results for those field IDs will be populated from a single call rather than making a driver call for each fieldId.

Parameters

- **device** – The device handle of the GPU to request field values for
- **valuesCount** – Number of entries in values that should be retrieved
- **values** – Array of *valuesCount* structures to hold field values. Each value's fieldId must be populated prior to this call

Returns

- *NVML_SUCCESS* if any values in *values* were populated. Note that you must check the nvmlReturn field of each value for each individual status
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *values* is NULL

5.2.1.18 Frame Buffer Capture Structures

5.2.1.18.1 Macros

NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_INFINITE
Bit specifying if capture was requested as blocking call.

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

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

NVML_NVFBC_SESSION_FLAG_CLASSIFICATIONMAP_ENABLED
Bit specifying classification map state.

NVML_NVFBC_SESSION_FLAG_DIFFMAP_ENABLED
Bit specifying differential map state.

5.2.1.18.2 Enumerations

nvmlFBCSessionType_t
Represents frame buffer capture session type.

5.2.1.18.3 Structs

nvmlFBCSessionInfo_t
Structure to hold FBC session data.

nvmlFBCStats_t
Structure to hold frame buffer capture sessions stats.

5.2.1.18.4 Macros

NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_INFINITE
Bit specifying if capture was requested as blocking call.

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

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

NVML_NVFBC_SESSION_FLAG_CLASSIFICATIONMAP_ENABLED
Bit specifying classification map state.

NVML_NVFBC_SESSION_FLAG_DIFFMAP_ENABLED
Bit specifying differential map state.

5.2.1.18.5 Enumerations

enum **nvmlFBCSessionType_t**
Represents frame buffer capture session type.
Values:

enumerator **NVML_FBC_SESSION_TYPE_UNKNOWN**
Unknown.

enumerator **NVML_FBC_SESSION_TYPE_TOSYS**
ToSys.

enumerator **NVML_FBC_SESSION_TYPE_CUDA**
Cuda.

enumerator **NVML_FBC_SESSION_TYPE_VID**
Vid.

enumerator **NVML_FBC_SESSION_TYPE_HWENC**
HEnc.

5.2.1.19 Initialization and Cleanup

NVML Methods that handle the NVML Library initialization and cleanup.

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

5.2.1.19.1 Macros

NVML_INIT_FLAG_FORCE_INIT
Force device initialization even when a previous nvmlInit was called with the NO_GPUS and NO_ATTACH flags.

NVML_INIT_FLAG_NO_ATTACH
Initialize the NVML Library without attaching any discovered devices.

NVML_INIT_FLAG_NO_GPUS
Initialize the NVML Library even when no devices are found.

5.2.1.19.2 Functions

nvmlReturn_t *nvmlInitWithFlags*(unsigned int flags)
Initialize the NVML Library lazily, without allocating any device state, with additional init flags.

nvmlReturn_t *nvmlInit_v2*(void)
Initialize the NVML Library lazily, without allocating any device state.

nvmlReturn_t *nvmlShutdown*(void)
Shut down and cleanup NVML Library state.

5.2.1.19.3 Macros

NVML_INIT_FLAG_FORCE_INIT
Force device initialization even when a previous nvmlInit was called with the NO_GPUS and NO_ATTACH flags.

NVML_INIT_FLAG_NO_ATTACH
Initialize the NVML Library without attaching any discovered devices.

NVML_INIT_FLAG_NO_GPUS
Initialize the NVML Library even when no devices are found.

5.2.1.19.4 Functions

nvmlReturn_t **nvmlInitWithFlags**(unsigned int flags)

Initialize the NVML Library lazily, without allocating any device state, with additional init flags.

A variant of *nvmlInit_v2()*, this will initialize the NVML Library state without enumerating any discovered devices. An option to pass in additional flags is provided to modify the behavior of NVML Library init. The usage of these flags can be obtained from NVML_INIT_FLAG_*. These flags can be combined together.

Other than the "flags" parameter, this method is completely identical to *nvmlInit_v2()*.

For all products.

See also

nvmlShutdown()

Note

With the environment variable `NVML_MPS_DEVICE_ENUMERATION` set to `1`, this API selects the MPS locality-domain view. *NVML_INIT_FLAG_NO_ATTACH* is not supported in that view. See *MPS Locality Domain Device Enumeration*.

Note

A reference count of the number of initializations is maintained, and a corresponding call to *nvmlShutdown()* needs to be issued once usage of the NVML Library is complete. Shutdown will only occur after the reference count reaches zero.

Parameters

flags – **[in]** NVML_INIT_FLAG_* flags that can modify NVML Init behavior.

Returns

- *NVML_SUCCESS* if the NVML Library was properly initialized.
- *NVML_ERROR_DRIVER_NOT_LOADED* if the NVIDIA driver is not running.
- *NVML_ERROR_NO_PERMISSION* if the NVML Library does not have permission to talk to the driver.
- *NVML_ERROR_UNKNOWN* if there is an unexpected error.

nvmlReturn_t **nvmlInit_v2**(void)

Initialize the NVML Library lazily, without allocating any device state.

This will initialize the NVML Library state without enumerating any discovered devices. This will allow NVML to communicate with a device, even if other devices are in an unstable or bad state. Enumeration of a device can be done by obtaining the device handle via the nvmlDeviceGetHandleBy* class of APIs.

This method needs to be called once before any usage of NVML Library APIs.

For all products.

See also

nvmlShutdown()

Note

With the environment variable `NVML_MPS_DEVICE_ENUMERATION` set to `1`, this API selects the MPS locality-domain view described in *MPS Locality Domain Device Enumeration*.

Note

A reference count of the number of initializations is maintained, and a corresponding call to *nvmlShutdown()* needs to be issued once usage of the NVML Library is complete. Shutdown will only occur after the reference count reaches zero.

Returns

- *NVML_SUCCESS* if the NVML Library was properly initialized.
- *NVML_ERROR_DRIVER_NOT_LOADED* if the NVIDIA driver is not running.
- *NVML_ERROR_NO_PERMISSION* if the NVML Library does not have permission to talk to the driver.
- *NVML_ERROR_UNKNOWN* if there is an unexpected error.

nvmlReturn_t **nvmlShutdown**(void)

Shut down and cleanup NVML Library state.

This will shut down and cleanup NVML Library state by releasing all device and library resources previously allocated with *nvmlInit_v2()* or *nvmlInitWithFlags()*. This should be called after all NVML work is done, and once for each call to *nvmlInit_v2()* or *nvmlInitWithFlags()*.

Complete shutdown will only occur when the reference count of all prior NVML initializations reaches zero. No error will be reported if this is called more times than *nvmlInit_v2()* or *nvmlInitWithFlags()*.

For all products.

Note

The final matching shutdown releases the process-local MPS locality-domain view described in *MPS Locality Domain Device Enumeration*.

Returns

- *NVML_SUCCESS* if the NVML Library was properly shut down.
- *NVML_ERROR_UNINITIALIZED* if the NVML Library was not previously initialized.

- *NVML_ERROR_UNKNOWN* if there is an unexpected error.

5.2.1.20 MPS Locality Domain Device Enumeration

On supported Linux systems, set the following environment variable before the first NVML initialization to enumerate the device view reported by MPS:

```
NVML_MPS_DEVICE_ENUMERATION=1
```

Each MPS locality domain has an `nvmlDevice_t` handle and an NVML index in this process-local view. A device that is not divided into locality domains can appear as one MPS device entry. These handles do not represent physical parent GPUs, and unsupported operations are not redirected to a parent.

The supported APIs are:

- Initialization and library information: `nvmlInit()`, *nvmlInit_v2()*, *nvmlInitWithFlags()*, *nvmlShutdown()*, *nvmlSystemGetNVMLVersion()*, and *nvmlErrorString()*.
- Enumeration and identity: *nvmlDeviceGetCount_v2()*, *nvmlDeviceGetHandleByIndex_v2()*, *nvmlDeviceGetIndex()*, *nvmlDeviceGetHandleByUUID()*, *nvmlDeviceGetUUID()*, *nvmlDeviceGetHandleByPciBusId_v2()*, and *nvmlDeviceGetName()*.
- Device attributes: *nvmlDeviceGetCudaComputeCapability()*, `nvmlDeviceGetPciInfo()`, `nvmlDeviceGetPciInfo_v2()`, *nvmlDeviceGetPciInfo_v3()*, *nvmlDeviceGetNumaNodeId()*, and *nvmlDeviceGetCpuAffinity()*.
- Telemetry and capability status: *nvmlDeviceGetMemoryInfo()*, *nvmlDeviceGetMemoryInfo_v2()*, *nvmlDeviceGetP2PStatus()*, and *nvmlDeviceGetAccountingMode()*.

Every other public NVML API returns *NVML_ERROR_NOT_SUPPORTED* while this view is selected. *NVML_INIT_FLAG_NO_ATTACH* and fork after initialization are unsupported. If the selected MPS session becomes invalid, MPS-backed APIs return *NVML_ERROR_INVALID_STATE* without falling back to physical GPU enumeration.

5.2.1.21 Multi Instance GPU Management

This chapter describes NVML operations that are associated with Multi Instance GPU management.

5.2.1.21.1 Macros

NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_COUNT
Number of engine profiles.

NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_SHARED
All the engines except multiprocessors would be shared.

NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE
Compute instance profiles.

NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE_REV1
1_SLICE compute instance profile (rev1).

NVML_COMPUTE_INSTANCE_PROFILE_2_SLICE
2_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_3_SLICE
3_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_4_SLICE
4_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_6_SLICE
6_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_7_SLICE
7_SLICE compute instance profile (perf optimized for host work scheduling).

NVML_COMPUTE_INSTANCE_PROFILE_7_SLICE_NVL
7_SLICE compute instance profile (perf optimized for multi-GPU use).

NVML_COMPUTE_INSTANCE_PROFILE_8_SLICE
8_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_CAPS_GFX
MIG compute instance profile capability.

NVML_COMPUTE_INSTANCE_PROFILE_COUNT
Number of compute instance profiles.

NVML_DEVICE_MIG_DISABLE
Disable Multi Instance GPU mode.

NVML_DEVICE_MIG_ENABLE
Enable Multi Instance GPU mode.

NVML_GPU_INSTANCE_PROFILE_1_SLICE
GPU instance profiles.

NVML_GPU_INSTANCE_PROFILE_1_SLICE_ALL_ME
1_SLICE profile with all of GPU Decode, Encoder, JPEG, OFA engines.

NVML_GPU_INSTANCE_PROFILE_1_SLICE_GFX
1_SLICE gfx capable profile.

NVML_GPU_INSTANCE_PROFILE_1_SLICE_NO_ME
1_SLICE profile with none of Decode, Encoder, JPEG, OFA engines.

NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV1
1_SLICE GPU instance profile (rev1).

NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV2
1_SLICE GPU instance profile (rev2).

NVML_GPU_INSTANCE_PROFILE_2_SLICE
2_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_2_SLICE_ALL_ME

NVML_GPU_INSTANCE_PROFILE_2_SLICE_GFX
2_SLICE gfx capable profile.

NVML_GPU_INSTANCE_PROFILE_2_SLICE_NO_ME
2_SLICE profile with none of Decode, Encoder, JPEG, OFA engines.

NVML_GPU_INSTANCE_PROFILE_2_SLICE_REV1
2_SLICE GPU instance profile (rev1).

NVML_GPU_INSTANCE_PROFILE_3_SLICE
3_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_3_SLICE_GFX
3_SLICE gfx + media capable profile.

NVML_GPU_INSTANCE_PROFILE_4_SLICE
4_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_4_SLICE_GFX
4_SLICE gfx capable profile.

NVML_GPU_INSTANCE_PROFILE_6_SLICE
6_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_7_SLICE
7_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_8_SLICE
8_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_CAPS_GFX
GFX support.

NVML_GPU_INSTANCE_PROFILE_CAPS_P2P
MIG GPU instance profile capability.

NVML_GPU_INSTANCE_PROFILE_COUNT
Total number of GPU instance profiles.

NVML_GPU_INTSTANCE_PROFILE_CAPS_P2P
Deprecated, do not use.

nvmlComputeInstanceProfileInfo_v2
Version identifier value for *nvmlComputeInstanceProfileInfo_v2_t::version* .

nvmlComputeInstanceProfileInfo_v3
Version identifier value for *nvmlComputeInstanceProfileInfo_v3_t::version* .

nvmlGpuInstanceProfileInfo_v2
Version identifier value for *nvmlGpuInstanceProfileInfo_v2_t::version* .

nvmlGpuInstanceProfileInfo_v3
Version identifier value for *nvmlGpuInstanceProfileInfo_v3_t::version* .

5.2.1.21.2 Functions

nvmlReturn_t *nvmlComputeInstanceDestroy*(nvmlComputeInstance_t computeInstance)
Destroy compute instance.

nvmlReturn_t *nvmlComputeInstanceGetInfo_v2*(nvmlComputeInstance_t computeInstance, nvmlComputeInstanceInfo_t *info)
Get compute instance information.

nvmlReturn_t *nvmlDeviceCreateGpuInstance*(nvmlDevice_t device, unsigned int profileId, nvmlGpuInstance_t *gpuInstance)
Create GPU instance.

nvmlReturn_t *nvmlDeviceCreateGpuInstanceWithPlacement*(nvmlDevice_t device, unsigned int profileId, const nvmlGpuInstancePlacement_t *placement, nvmlGpuInstance_t *gpuInstance)
Create GPU instance with the specified placement.

nvmlReturn_t *nvmlDeviceGetComputeInstanceId*(nvmlDevice_t device, unsigned int *id)
Get compute instance ID for the given MIG device handle.

nvmlReturn_t *nvmlDeviceGetDeviceHandleFromMigDeviceHandle*(nvmlDevice_t migDevice, nvmlDevice_t *device)
Get parent device handle from a MIG device handle.

nvmlReturn_t *nvmlDeviceGetGpuInstanceById*(nvmlDevice_t device, unsigned int id, nvmlGpuInstance_t *gpuInstance)
Get GPU instances for given instance ID.

nvmlReturn_t *nvmlDeviceGetGpuInstanceId*(nvmlDevice_t device, unsigned int *id)
Get GPU instance ID for the given MIG device handle.

nvmlReturn_t *nvmlDeviceGetGpuInstancePossiblePlacements_v2*(nvmlDevice_t device, unsigned int profileId, nvmlGpuInstancePlacement_t *placements, unsigned int *count)
Get GPU instance placements.

nvmlReturn_t *nvmlDeviceGetGpuInstanceProfileInfo*(nvmlDevice_t device, unsigned int profile, nvmlGpuInstanceProfileInfo_t *info)
Get GPU instance profile information.

nvmlReturn_t *nvmlDeviceGetGpuInstanceProfileInfoByIdV*(nvmlDevice_t device, unsigned int profileId, nvmlGpuInstanceProfileInfo_v2_t *info)
GPU instance profile query function that accepts profile ID, instead of profile name.

nvmlReturn_t *nvmlDeviceGetGpuInstanceProfileInfoV*(nvmlDevice_t device, unsigned int profile, nvmlGpuInstanceProfileInfo_v2_t *info)
Versioned wrapper around *nvmlDeviceGetGpuInstanceProfileInfo* that accepts a versioned *nvmlGpuInstanceProfileInfo_v2_t* or later output structure.

nvmlReturn_t *nvmlDeviceGetGpuInstanceRemainingCapacity*(nvmlDevice_t device, unsigned int profileId, unsigned int *count)
Get GPU instance profile capacity.

nvmlReturn_t *nvmlDeviceGetGpuInstances*(nvmlDevice_t device, unsigned int profileId, nvmlGpuInstance_t *gpuInstances, unsigned int *count)
Get GPU instances for given profile ID.

nvmlReturn_t *nvmlDeviceGetMaxMigDeviceCount*(nvmlDevice_t device, unsigned int *count)
Get the maximum number of MIG devices that can exist under a given parent NVML device.

nvmlReturn_t *nvmlDeviceGetMigDeviceHandleByIndex*(nvmlDevice_t device, unsigned int index, nvmlDevice_t *migDevice)
Get MIG device handle for the given index under its parent NVML device.

nvmlReturn_t *nvmlDeviceGetMigMode*(nvmlDevice_t device, unsigned int *currentMode, unsigned int *pendingMode)
Get MIG mode for the device.

nvmlReturn_t *nvmlDeviceIsMigDeviceHandle*(nvmlDevice_t device, unsigned int *isMigDevice)
Test if the given handle refers to a MIG device.

nvmlReturn_t *nvmlDeviceSetMigMode*(nvmlDevice_t device, unsigned int mode, nvmlReturn_t *activationStatus)
Set MIG mode for the device.

nvmlReturn_t *nvmlGpuInstanceCreateComputeInstance*(nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstance_t *computeInstance)
Create compute instance.

nvmlReturn_t *nvmlGpuInstanceCreateComputeInstanceWithPlacement*(nvmlGpuInstance_t gpuInstance, unsigned int profileId, const nvmlComputeInstancePlacement_t *placement, nvmlComputeInstance_t *computeInstance)
Create compute instance with the specified placement.

nvmlReturn_t *nvmlGpuInstanceDestroy*(nvmlGpuInstance_t gpuInstance)
Destroy GPU instance.

nvmlReturn_t *nvmlGpuInstanceGetComputeInstanceById*(nvmlGpuInstance_t gpuInstance, unsigned int id, nvmlComputeInstance_t *computeInstance)
Get compute instance for given instance ID.

nvmlReturn_t *nvmlGpuInstanceGetComputeInstancePossiblePlacements*(nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstancePlacement_t *placements, unsigned int *count)
Get compute instance placements.

nvmlReturn_t *nvmlGpuInstanceGetComputeInstanceProfileInfo*(nvmlGpuInstance_t gpuInstance, unsigned int profile, unsigned int engProfile, nvmlComputeInstanceProfileInfo_t *info)
Get compute instance profile information.

nvmlReturn_t *nvmlGpuInstanceGetComputeInstanceProfileInfoV*(nvmlGpuInstance_t gpuInstance, unsigned int profile, unsigned int engProfile, nvmlComputeInstanceProfileInfo_v2_t *info)
Versioned wrapper around *nvmlGpuInstanceGetComputeInstanceProfileInfo* that accepts a versioned *nvmlComputeInstanceProfileInfo_v2_t* or later output structure.

nvmlReturn_t *nvmlGpuInstanceGetComputeInstanceRemainingCapacity*(nvmlGpuInstance_t gpuInstance, unsigned int profileId, unsigned int *count)
Get compute instance profile capacity.

nvmlReturn_t *nvmlGpuInstanceGetComputeInstances*(nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstance_t *computeInstances, unsigned int *count)
Get compute instances for given profile ID.

nvmlReturn_t *nvmlGpuInstanceGetInfo*(nvmlGpuInstance_t gpuInstance, nvmlGpuInstanceInfo_t *info)
Get GPU instance information.

5.2.1.21.3 Structs

nvmlComputeInstanceInfo_t

nvmlComputeInstancePlacement_t

nvmlComputeInstanceProfileInfo_t
Compute instance profile information.

nvmlComputeInstanceProfileInfo_v2_t
Compute instance profile information (v2).

nvmlComputeInstanceProfileInfo_v3_t
Compute instance profile information (v3).

nvmlGpuInstanceInfo_t

nvmlGpuInstancePlacement_t

nvmlGpuInstanceProfileInfo_t
GPU instance profile information.

nvmlGpuInstanceProfileInfo_v2_t
GPU instance profile information (v2).

nvmlGpuInstanceProfileInfo_v3_t
GPU instance profile information (v3).

5.2.1.21.4 Typedefs

nvmlComputeInstance_t

5.2.1.21.5 Macros

NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_COUNT
Number of engine profiles.

NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_SHARED
All the engines except multiprocessors would be shared.

NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE
Compute instance profiles.

These macros should be passed to *nvmlGpuInstanceGetComputeInstanceProfileInfo* to retrieve the detailed information about a compute instance such as profile ID, engine counts 1_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE_REV1
1_SLICE compute instance profile (rev1).

NVML_COMPUTE_INSTANCE_PROFILE_2_SLICE
2_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_3_SLICE
3_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_4_SLICE
4_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_6_SLICE
6_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_7_SLICE
7_SLICE compute instance profile (perf optimized for host work scheduling).

NVML_COMPUTE_INSTANCE_PROFILE_7_SLICE_NVL
7_SLICE compute instance profile (perf optimized for multi-GPU use).

NVML_COMPUTE_INSTANCE_PROFILE_8_SLICE
8_SLICE compute instance profile.

NVML_COMPUTE_INSTANCE_PROFILE_CAPS_GFX

MIG compute instance profile capability.

Bit field values representing MIG profile capabilities *nvmlComputeInstanceProfileInfo_v3_t::capabilities*

NVML_COMPUTE_INSTANCE_PROFILE_COUNT

Number of compute instance profiles.

NVML_DEVICE_MIG_DISABLE

Disable Multi Instance GPU mode.

Disable Multi Instance GPU mode.

NVML_DEVICE_MIG_ENABLE

Enable Multi Instance GPU mode.

Enable Multi Instance GPU mode.

NVML_GPU_INSTANCE_PROFILE_1_SLICE

GPU instance profiles.

These macros should be passed to *nvmlDeviceGetGpuInstanceProfileInfo* to retrieve the detailed information about a GPU instance such as profile ID, engine counts. 1_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_1_SLICE_ALL_ME

1_SLICE profile with all of GPU Decode, Encoder, JPEG, OFA engines.

NVML_GPU_INSTANCE_PROFILE_1_SLICE_GFX

1_SLICE gfx capable profile.

NVML_GPU_INSTANCE_PROFILE_1_SLICE_NO_ME

1_SLICE profile with none of Decode, Encoder, JPEG, OFA engines.

NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV1

1_SLICE GPU instance profile (rev1).

NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV2

1_SLICE GPU instance profile (rev2).

NVML_GPU_INSTANCE_PROFILE_2_SLICE

2_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_2_SLICE_ALL_ME

NVML_GPU_INSTANCE_PROFILE_2_SLICE_GFX

2_SLICE gfx capable profile.

NVML_GPU_INSTANCE_PROFILE_2_SLICE_NO_ME

2_SLICE profile with none of Decode, Encoder, JPEG, OFA engines.

NVML_GPU_INSTANCE_PROFILE_2_SLICE_REV1

2_SLICE GPU instance profile (rev1).

NVML_GPU_INSTANCE_PROFILE_3_SLICE

3_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_3_SLICE_GFX

3_SLICE gfx + media capable profile.

NVML_GPU_INSTANCE_PROFILE_4_SLICE

4_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_4_SLICE_GFX

4_SLICE gfx capable profile.

NVML_GPU_INSTANCE_PROFILE_6_SLICE

6_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_7_SLICE

7_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_8_SLICE

8_SLICE GPU instance profile.

NVML_GPU_INSTANCE_PROFILE_CAPS_GFX

GFX support.

NVML_GPU_INSTANCE_PROFILE_CAPS_P2P

MIG GPU instance profile capability.

Bit field values representing MIG profile capabilities *nvmlGpuInstanceProfileInfo_v3_t::capabilities*
Peer-to-Peer support.

NVML_GPU_INSTANCE_PROFILE_COUNT

Total number of GPU instance profiles.

NVML_GPU_INTSTANCE_PROFILE_CAPS_P2P

Deprecated, do not use.

nvmlComputeInstanceProfileInfo_v2

Version identifier value for *nvmlComputeInstanceProfileInfo_v2_t::version*.

nvmlComputeInstanceProfileInfo_v3

Version identifier value for *nvmlComputeInstanceProfileInfo_v3_t::version*.

nvmlGpuInstanceProfileInfo_v2

Version identifier value for *nvmlGpuInstanceProfileInfo_v2_t::version*.

nvmlGpuInstanceProfileInfo_v3

Version identifier value for *nvmlGpuInstanceProfileInfo_v3_t::version*.

5.2.1.21.6 Functions

nvmlReturn_t **nvmlComputeInstanceDestroy**(
nvmlComputeInstance_t computeInstance,
)

Destroy compute instance.

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

Parameters

computeInstance – The compute instance handle

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *computeInstance* is invalid
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_IN_USE* If the compute instance is in use. This error would be returned if processes (e.g. CUDA application) are active on the compute instance.

nvmlReturn_t **nvmlComputeInstanceGetInfo_v2**(
nvmlComputeInstance_t computeInstance, *nvmlComputeInstanceInfo_t* *info,
)

Get compute instance information.

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

Parameters

- **computeInstance** – The compute instance handle
- **info** – Return compute instance information

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *computeInstance* or *info* are invalid

- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlDeviceCreateGpuInstance**(
nvmlDevice_t device, unsigned int profileId, *nvmlGpuInstance_t* *gpuInstance,
)

Create GPU instance.

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

If the parent device is unbound, reset or the GPU instance is destroyed explicitly, the GPU instance handle would become invalid. The GPU instance must be recreated to acquire a valid handle.

Parameters

- **device** – The identifier of the target device
- **profileId** – The GPU instance profile ID. See *nvmlDeviceGetGpuInstanceProfileInfo*
- **gpuInstance** – Returns the GPU instance handle

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *profile*, *profileId* or *gpuInstance* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled or in vGPU guest
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_INSUFFICIENT_RESOURCES* If the requested GPU instance could not be created

nvmlReturn_t **nvmlDeviceCreateGpuInstanceWithPlacement**(
nvmlDevice_t device, unsigned int profileId, const *nvmlGpuInstancePlacement_t* *placement, *nvmlGpuInstance_t* *gpuInstance,
)

Create GPU instance with the specified placement.

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

If the parent device is unbound, reset or the GPU instance is destroyed explicitly, the GPU instance handle would become invalid. The GPU instance must be recreated to acquire a valid handle.

Parameters

- **device** – The identifier of the target device
- **profileId** – The GPU instance profile ID. See *nvmlDeviceGetGpuInstanceProfileInfo*

- **placement** – The requested placement. See *nvmlDeviceGetGpuInstancePossiblePlacements_v2*
- **gpuInstance** – Returns the GPU instance handle

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *profile*, *profileId*, *placement* or *gpuInstance* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled or in vGPU guest
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_INSUFFICIENT_RESOURCES* If the requested GPU instance could not be created

nvmlReturn_t **nvmlDeviceGetComputeInstanceId**(
    nvmlDevice_t device, unsigned int *id,
)

Get compute instance ID for the given MIG device handle.

Compute instance IDs are unique per GPU instance and remain valid until the compute instance is destroyed.

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

Parameters

- **device** – Target MIG device handle
- **id** – Compute instance ID

Returns

- *NVML_SUCCESS* if instance ID was successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *id* reference is invalid
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetDeviceHandleFromMigDeviceHandle**(
    nvmlDevice_t migDevice, *nvmlDevice_t* *device,
)

Get parent device handle from a MIG device handle.

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

Parameters

- **migDevice** – MIG device handle

- **device** – Device handle

Returns

- *NVML_SUCCESS* if *device* handle was successfully created
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *migDevice* or *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetGpuInstanceById**(
nvmlDevice_t device, unsigned int id, *nvmlGpuInstance_t* *gpuInstance,
)

Get GPU instances for given instance ID.

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

Parameters

- **device** – The identifier of the target device
- **id** – The GPU instance ID
- **gpuInstance** – Returns GPU instance

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *id* or *gpuInstance* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_NOT_FOUND* If the GPU instance is not found.

nvmlReturn_t **nvmlDeviceGetGpuInstanceId**(
nvmlDevice_t device, unsigned int *id,
)

Get GPU instance ID for the given MIG device handle.

GPU instance IDs are unique per device and remain valid until the GPU instance is destroyed.

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

Parameters

- **device** – Target MIG device handle
- **id** – GPU instance ID

Returns

- *NVML_SUCCESS* if instance ID was successfully retrieved

- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *id* reference is invalid
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetGpuInstancePossiblePlacements_v2**(
nvmlDevice_t device, unsigned int profileId, *nvmlGpuInstancePlacement_t* *placements, unsigned int *count,
)

Get GPU instance placements.

A placement represents the location of a GPU instance within a device. This API only returns all the possible placements for the given profile regardless of whether MIG is enabled or not. A created GPU instance occupies memory slices described by its placement. Creation of new GPU instance will fail if there is overlap with the already occupied memory slices.

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

Parameters

- **device** – The identifier of the target device
- **profileId** – The GPU instance profile ID. See *nvmlDeviceGetGpuInstanceProfileInfo*
- **placements** – Returns placements allowed for the profile. Can be NULL to discover number of allowed placements for this profile. If non-NULL must be large enough to accommodate the placements supported by the profile.
- **count** – Returns number of allowed placemenets for the profile.

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *profileId* or *count* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't support MIG or *profileId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlDeviceGetGpuInstanceProfileInfo**(
nvmlDevice_t device, unsigned int profile, *nvmlGpuInstanceProfileInfo_t* *info,
)

Get GPU instance profile information.

Information provided by this API is immutable throughout the lifetime of a MIG mode.

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

Note

This API can be used to enumerate all MIG profiles supported by NVML in a forward compatible way by invoking it on *profile* values starting from 0, until the API returns *NVML_ERROR_INVALID_ARGUMENT*.

Parameters

- **device** – The identifier of the target device
- **profile** – One of the NVML_GPU_INSTANCE_PROFILE_*
- **info** – Returns detailed profile information

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *profile* or *info* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't support MIG or *profile* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlDeviceGetGpuInstanceProfileInfoByIdV**(
    nvmlDevice_t device, unsigned int profileId, *nvmlGpuInstanceProfileInfo_v2_t* *info,
)

GPU instance profile query function that accepts profile ID, instead of profile name.

It accepts a versioned *nvmlGpuInstanceProfileInfo_v2_t* or later output structure.

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

Note

The caller must set the *nvmlGpuInstanceProfileInfo_v2_t::version* field to the appropriate version prior to calling this function. For example:

```
nvmlGpuInstanceProfileInfo_v2_t profileInfo =
    { .version = nvmlGpuInstanceProfileInfo_v2 };
nvmlReturn_t result = nvmlDeviceGetGpuInstanceProfileInfoV(device,
                                                           profile,
                                                           &
↪profileInfo);
```

Parameters

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

- **profileId** – One of the profile IDs.
- **info** – Returns detailed profile information

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *profileId*, *info*, or *info->version* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled or *profile* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlDeviceGetGpuInstanceProfileInfoV**(
nvmlDevice_t device, unsigned int profile, *nvmlGpuInstanceProfileInfo_v2_t* *info,
)

Versioned wrapper around *nvmlDeviceGetGpuInstanceProfileInfo* that accepts a versioned *nvmlGpuInstanceProfileInfo_v2_t* or later output structure.

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

Note

The caller must set the *nvmlGpuInstanceProfileInfo_v2_t::version* field to the appropriate version prior to calling this function. For example:

```
nvmlGpuInstanceProfileInfo_v2_t profileInfo =
    { .version = nvmlGpuInstanceProfileInfo_v2 };
nvmlReturn_t result = nvmlDeviceGetGpuInstanceProfileInfoV(device,
                                                           profile,
                                                           &
↪profileInfo);
```

Parameters

- **device** – The identifier of the target device
- **profile** – One of the NVML_GPU_INSTANCE_PROFILE_*
- **info** – Returns detailed profile information

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *profile*, *info*, or *info->version* are invalid

- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled or *profile* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlDeviceGetGpuInstanceRemainingCapacity**(
nvmlDevice_t device, unsigned int profileId, unsigned int *count,
)

Get GPU instance profile capacity.

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

Parameters

- **device** – The identifier of the target device
- **profileId** – The GPU instance profile ID. See *nvmlDeviceGetGpuInstanceProfileInfo*
- **count** – Returns remaining instance count for the profile ID

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *profileId* or *count* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled or *profileId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlDeviceGetGpuInstances**(
nvmlDevice_t device, unsigned int profileId, *nvmlGpuInstance_t* *gpuInstances, unsigned int *count,
)

Get GPU instances for given profile ID.

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

Parameters

- **device** – The identifier of the target device
- **profileId** – The GPU instance profile ID. See *nvmlDeviceGetGpuInstanceProfileInfo*
- **gpuInstances** – Returns pre-exiting GPU instances, the buffer must be large enough to accommodate the instances supported by the profile. See *nvmlDeviceGetGpuInstanceProfileInfo*
- **count** – The count of returned GPU instances

Returns

- *NVML_SUCCESS* Upon success

- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *profileId*, *gpuInstances* or *count* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlDeviceGetMaxMigDeviceCount**(
    nvmlDevice_t device, unsigned int *count,
)

Get the maximum number of MIG devices that can exist under a given parent NVML device.

Returns zero if MIG is not supported or enabled.

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

Parameters

- **device** – Target device handle
- **count** – Count of MIG devices

Returns

- *NVML_SUCCESS* if *count* was successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *count* reference is invalid
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetMigDeviceHandleByIndex**(
    nvmlDevice_t device, unsigned int index, *nvmlDevice_t* *migDevice,
)

Get MIG device handle for the given index under its parent NVML device.

If the compute instance is destroyed either explicitly or by destroying, resetting or unbinding the parent GPU instance or the GPU device itself the MIG device handle would remain invalid and must be requested again using this API. Handles may be reused and their properties can change in the process.

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

Parameters

- **device** – Reference to the parent GPU device handle
- **index** – Index of the MIG device
- **migDevice** – Reference to the MIG device handle

Returns

- *NVML_SUCCESS* if *migDevice* handle was successfully created
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized

- *NVML_ERROR_INVALID_ARGUMENT* if *device*, *index* or *migDevice* reference is invalid
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device
- *NVML_ERROR_NOT_FOUND* if no valid MIG device was found at *index*
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetMigMode**(
nvmlDevice_t device, unsigned int *currentMode, unsigned int *pendingMode,
)

Get MIG mode for the device.

For Ampere or newer fully supported devices.

Changing MIG modes may require device unbind or reset. The "pending" MIG mode refers to the target mode following the next activation trigger.

Parameters

- **device** – The identifier of the target device
- **currentMode** – Returns the current mode, *NVML_DEVICE_MIG_DISABLE* or *NVML_DEVICE_MIG_ENABLE*
- **pendingMode** – Returns the pending mode, *NVML_DEVICE_MIG_DISABLE* or *NVML_DEVICE_MIG_ENABLE*

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *currentMode* or *pendingMode* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't support MIG mode
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceIsMigDeviceHandle**(
nvmlDevice_t device, unsigned int *isMigDevice,
)

Test if the given handle refers to a MIG device.

A MIG device handle is an NVML abstraction which maps to a MIG compute instance. These overloaded references can be used (with some restrictions) interchangeably with a GPU device handle to execute queries at a per-compute instance granularity.

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

Parameters

- **device** – NVML handle to test
- **isMigDevice** – True when handle refers to a MIG device

Returns

- *NVML_SUCCESS* if *device* status was successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* handle or *isMigDevice* reference is invalid
- *NVML_ERROR_NOT_SUPPORTED* if this check is not supported by the device
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceSetMigMode**(
nvmlDevice_t device, unsigned int mode, *nvmlReturn_t* *activationStatus,
)

Set MIG mode for the device.

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

This mode determines whether a GPU instance can be created.

This API may unbind or reset the device to activate the requested mode. Thus, the attributes associated with the device, such as minor number, might change. The caller of this API is expected to query such attributes again.

On certain platforms like pass-through virtualization, where reset functionality may not be exposed directly, VM reboot is required. *activationStatus* would return *NVML_ERROR_RESET_REQUIRED* for such cases.

activationStatus would return the appropriate error code upon unsuccessful activation. For example, if device unbind fails because the device isn't idle, *NVML_ERROR_IN_USE* would be returned. The caller of this API is expected to idle the device and retry setting the *mode*.

Note

On Windows, only disabling MIG mode is supported. *activationStatus* would return *NVML_ERROR_NOT_SUPPORTED* as GPU reset is not supported on Windows through this API.

Parameters

- **device** – The identifier of the target device
- **mode** – The mode to be set, *NVML_DEVICE_MIG_DISABLE* or *NVML_DEVICE_MIG_ENABLE*
- **activationStatus** – The activationStatus status

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*,*mode* or *activationStatus* are invalid
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't support MIG mode

nvmlReturn_t **nvmlGpuInstanceCreateComputeInstance**(
nvmlGpuInstance_t gpuInstance, unsigned int profileId, *nvmlComputeInstance_t* *computeInstance,
)

Create compute instance.

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

If the parent device is unbound, reset or the parent GPU instance is destroyed or the compute instance is destroyed explicitly, the compute instance handle would become invalid. The compute instance must be recreated to acquire a valid handle.

Parameters

- **gpuInstance** – The identifier of the target GPU instance
- **profileId** – The compute instance profile ID. See *nvmlGpuInstanceGetComputeInstanceProfileInfo*
- **computeInstance** – Returns the compute instance handle

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance*, *profile*, *profileId* or *computeInstance* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *profileId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_INSUFFICIENT_RESOURCES* If the requested compute instance could not be created

nvmlReturn_t **nvmlGpuInstanceCreateComputeInstanceWithPlacement**(
nvmlGpuInstance_t gpuInstance, unsigned int profileId, const *nvmlComputeInstancePlacement_t* *placement, *nvmlComputeInstance_t* *computeInstance,
)

Create compute instance with the specified placement.

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

If the parent device is unbound, reset or the parent GPU instance is destroyed or the compute instance is destroyed explicitly, the compute instance handle would become invalid. The compute instance must be recreated to acquire a valid handle.

Parameters

- **gpuInstance** – The identifier of the target GPU instance
- **profileId** – The compute instance profile ID. See *nvmlGpuInstanceGetComputeInstanceProfileInfo*
- **placement** – The requested placement. See *nvmlGpuInstanceGetComputeInstancePossiblePlacements*
- **computeInstance** – Returns the compute instance handle

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance*, *profile*, *profileId* or *computeInstance* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *profileId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_INSUFFICIENT_RESOURCES* If the requested compute instance could not be created

nvmlReturn_t **nvmlGpuInstanceDestroy**(*nvmlGpuInstance_t* gpuInstance)

Destroy GPU instance.

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

Parameters

gpuInstance – The GPU instance handle

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled or in vGPU guest
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_IN_USE* If the GPU instance is in use. This error would be returned if processes (e.g. CUDA application) or compute instances are active on the GPU instance.

nvmlReturn_t **nvmlGpuInstanceGetComputeInstanceById**(
nvmlGpuInstance_t gpuInstance, unsigned int id, *nvmlComputeInstance_t* *computeInstance,
)

Get compute instance for given instance ID.

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

Parameters

- **gpuInstance** – The identifier of the target GPU instance
- **id** – The compute instance ID
- **computeInstance** – Returns compute instance

Returns

- *NVML_SUCCESS* Upon success

- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device*, *ID* or *computeInstance* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_NOT_FOUND* If the compute instance is not found.

nvmlReturn_t **nvmlGpuInstanceGetComputeInstancePossiblePlacements**(
nvmlGpuInstance_t gpuInstance, unsigned int profileId, *nvmlComputeInstancePlacement_t* *placements, unsigned int *count,
)

Get compute instance placements.

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

A placement represents the location of a compute instance within a GPU instance. This API only returns all the possible placements for the given profile. A created compute instance occupies compute slices described by its placement. Creation of new compute instance will fail if there is overlap with the already occupied compute slices.

Parameters

- **gpuInstance** – The identifier of the target GPU instance
- **profileId** – The compute instance profile ID. See *nvmlGpuInstanceGetComputeInstanceProfileInfo*
- **placements** – Returns placements allowed for the profile. Can be NULL to discover number of allowed placements for this profile. If non-NULL must be large enough to accommodate the placements supported by the profile.
- **count** – Returns number of allowed placemenets for the profile.

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance*, *profileId* or *count* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *device* doesn't have MIG mode enabled or *profileId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlGpuInstanceGetComputeInstanceProfileInfo**(
nvmlGpuInstance_t gpuInstance, unsigned int profile, unsigned int engProfile, *nvmlComputeInstanceProfileInfo_t* *info,
)

Get compute instance profile information.

Information provided by this API is immutable throughout the lifetime of a MIG mode.

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

Note

This API can be used to enumerate all MIG profiles supported by NVML in a forward compatible way by invoking it on *profile* values starting from 0, until the API returns *NVML_ERROR_INVALID_ARGUMENT*.

Parameters

- **gpuInstance** – The identifier of the target GPU instance
- **profile** – One of the NVML_COMPUTE_INSTANCE_PROFILE_*
- **engProfile** – One of the NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_*
- **info** – Returns detailed profile information

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance*, *profile*, *engProfile* or *info* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *profile* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlGpuInstanceGetComputeInstanceProfileInfoV**(
nvmlGpuInstance_t gpuInstance, unsigned int profile, unsigned int engProfile, *nvmlComputeInstanceProfileInfo_v2_t* *info,
)

Versioned wrapper around *nvmlGpuInstanceGetComputeInstanceProfileInfo* that accepts a versioned *nvmlComputeInstanceProfileInfo_v2_t* or later output structure.

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

Note

The caller must set the *nvmlGpuInstanceProfileInfo_v2_t::version* field to the appropriate version prior to calling this function. For example:

```
nvmlComputeInstanceProfileInfo_v2_t profileInfo =
    { .version = nvmlComputeInstanceProfileInfo_v2 };
nvmlReturn_t result =
↪nvmlGpuInstanceGetComputeInstanceProfileInfoV(gpuInstance,

↪profile,

↪engProfile,
                                                &

↪profileInfo);
```

Parameters

- **gpuInstance** – The identifier of the target GPU instance
- **profile** – One of the NVML_COMPUTE_INSTANCE_PROFILE_*
- **engProfile** – One of the NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_*
- **info** – Returns detailed profile information

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance*, *profile*, *engProfile*, *info*, or *info->version* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *profile* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlGpuInstanceGetComputeInstanceRemainingCapacity**(
nvmlGpuInstance_t gpuInstance, unsigned int profileId, unsigned int *count,
)

Get compute instance profile capacity.

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

Parameters

- **gpuInstance** – The identifier of the target GPU instance
- **profileId** – The compute instance profile ID. See *nvmlGpuInstanceGetComputeInstanceProfileInfo*
- **count** – Returns remaining instance count for the profile ID

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance*, *profileId* or *availableCount* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *profileId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlGpuInstanceGetComputeInstances**(
nvmlGpuInstance_t gpuInstance, unsigned int profileId, *nvmlComputeInstance_t* *computeInstances, unsigned int *count,
)

Get compute instances for given profile ID.

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

Parameters

- **gpuInstance** – The identifier of the target GPU instance
- **profileId** – The compute instance profile ID. See *nvmlGpuInstanceGetComputeInstanceProfileInfo*
- **computeInstances** – Returns pre-exiting compute instances, the buffer must be large enough to accommodate the instances supported by the profile. See *nvmlGpuInstanceGetComputeInstanceProfileInfo*
- **count** – The count of returned compute instances

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance*, *profileId*, *computeInstances* or *count* are invalid
- *NVML_ERROR_NOT_SUPPORTED* If *profileId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

nvmlReturn_t **nvmlGpuInstanceGetInfo**(
    nvmlGpuInstance_t gpuInstance, *nvmlGpuInstanceInfo_t* *info,
)

Get GPU instance information.

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

Parameters

- **gpuInstance** – The GPU instance handle
- **info** – Return GPU instance information

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* or *info* are invalid
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation

5.2.1.21.7 Typedefs

typedef struct nvmlComputeInstance_st ***nvmlComputeInstance_t**

5.2.1.22 NVML GPM

Note

For NVIDIA vGPU Software products

Note

GPM is supported only on MIG-backed vGPU profiles that are allocated all of the instance's frame buffer

5.2.1.22.1 GPM Enums

5.2.1.22.1.1 Enumerations

nvmlGpmMetricId_t

GPM Metric Identifiers.

5.2.1.22.1.2 Enumerations

enum **nvmlGpmMetricId_t**

GPM Metric Identifiers.

Values:

enumerator **NVML_GPM_METRIC_GRAPHICS_UTIL**

Percentage of time any compute/graphics app was active on the GPU. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_SM_UTIL**

Percentage of SMs that were busy. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_SM_OCCUPANCY**

Percentage of warps that were active vs theoretical maximum. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_INTEGER_UTIL**

Percentage of time the GPU's SMs were doing integer operations. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_ANY_TENSOR_UTIL**

Percentage of time the GPU's SMs were doing ANY tensor operations. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_DFMA_TENSOR_UTIL**

Percentage of time the GPU's SMs were doing DFMA tensor operations. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_HMMA_TENSOR_UTIL**

Percentage of time the GPU's SMs were doing HMMA tensor operations. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_DMMA_TENSOR_UTIL**
Percentage of time the GPU's SMs were doing DMMA tensor operations. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_IMMA_TENSOR_UTIL**
Percentage of time the GPU's SMs were doing IMMA tensor operations. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_DRAM_BW_UTIL**
Percentage of DRAM bw used vs theoretical maximum. 0.0 - 100.0 */.

enumerator **NVML_GPM_METRIC_FP64_UTIL**
Percentage of time the GPU's SMs were doing non-tensor FP64 math. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_FP32_UTIL**
Percentage of time the GPU's SMs were doing non-tensor FP32 math. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_FP16_UTIL**
Percentage of time the GPU's SMs were doing non-tensor FP16 math. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_PCIE_TX_PER_SEC**
PCIe traffic from this GPU in MiB/sec.

enumerator **NVML_GPM_METRIC_PCIE_RX_PER_SEC**
PCIe traffic to this GPU in MiB/sec.

enumerator **NVML_GPM_METRIC_NVDEC_0_UTIL**
Percent utilization of NVDEC 0. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVDEC_1_UTIL**
Percent utilization of NVDEC 1. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVDEC_2_UTIL**
Percent utilization of NVDEC 2. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVDEC_3_UTIL**
Percent utilization of NVDEC 3. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVDEC_4_UTIL**
Percent utilization of NVDEC 4. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVDEC_5_UTIL**
Percent utilization of NVDEC 5. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVDEC_6_UTIL**
Percent utilization of NVDEC 6. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVDEC_7_UTIL**
Percent utilization of NVDEC 7. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVJPG_0_UTIL**
Percent utilization of NVJPG 0. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVJPG_1_UTIL**
Percent utilization of NVJPG 1. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVJPG_2_UTIL**
Percent utilization of NVJPG 2. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVJPG_3_UTIL**
Percent utilization of NVJPG 3. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVJPG_4_UTIL**
Percent utilization of NVJPG 4. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVJPG_5_UTIL**
Percent utilization of NVJPG 5. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVJPG_6_UTIL**
Percent utilization of NVJPG 6. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVJPG_7_UTIL**
Percent utilization of NVJPG 7. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVOFA_0_UTIL**
Percent utilization of NVOFA 0. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVOFA_1_UTIL**
Percent utilization of NVOFA 1. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVLINK_TOTAL_RX_PER_SEC**
NvLink read bandwidth for all links in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_TOTAL_TX_PER_SEC**
NvLink write bandwidth for all links in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L0_RX_PER_SEC**
NvLink read bandwidth for link 0 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L0_TX_PER_SEC**
NvLink write bandwidth for link 0 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L1_RX_PER_SEC**
NvLink read bandwidth for link 1 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L1_TX_PER_SEC**
NvLink write bandwidth for link 1 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L2_RX_PER_SEC**
NvLink read bandwidth for link 2 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L2_TX_PER_SEC**
NvLink write bandwidth for link 2 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L3_RX_PER_SEC**
NvLink read bandwidth for link 3 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L3_TX_PER_SEC**
NvLink write bandwidth for link 3 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L4_RX_PER_SEC**
NvLink read bandwidth for link 4 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L4_TX_PER_SEC**
NvLink write bandwidth for link 4 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L5_RX_PER_SEC**
NvLink read bandwidth for link 5 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L5_TX_PER_SEC**
NvLink write bandwidth for link 5 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L6_RX_PER_SEC**
NvLink read bandwidth for link 6 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L6_TX_PER_SEC**
NvLink write bandwidth for link 6 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L7_RX_PER_SEC**
NvLink read bandwidth for link 7 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L7_TX_PER_SEC**
NvLink write bandwidth for link 7 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L8_RX_PER_SEC**
NvLink read bandwidth for link 8 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L8_TX_PER_SEC**
NvLink write bandwidth for link 8 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L9_RX_PER_SEC**
NvLink read bandwidth for link 9 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L9_TX_PER_SEC**
NvLink write bandwidth for link 9 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L10_RX_PER_SEC**
NvLink read bandwidth for link 10 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L10_TX_PER_SEC**
NvLink write bandwidth for link 10 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L11_RX_PER_SEC**
NvLink read bandwidth for link 11 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L11_TX_PER_SEC**
NvLink write bandwidth for link 11 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L12_RX_PER_SEC**
NvLink read bandwidth for link 12 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L12_TX_PER_SEC**
NvLink write bandwidth for link 12 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L13_RX_PER_SEC**
NvLink read bandwidth for link 13 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L13_TX_PER_SEC**
NvLink write bandwidth for link 13 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L14_RX_PER_SEC**
NvLink read bandwidth for link 14 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L14_TX_PER_SEC**
NvLink write bandwidth for link 14 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L15_RX_PER_SEC**
NvLink read bandwidth for link 15 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L15_TX_PER_SEC**
NvLink write bandwidth for link 15 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L16_RX_PER_SEC**
NvLink read bandwidth for link 16 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L16_TX_PER_SEC**
NvLink write bandwidth for link 16 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L17_RX_PER_SEC**
NvLink read bandwidth for link 17 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L17_TX_PER_SEC**
NvLink write bandwidth for link 17 in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_TOTAL_TX_PER_SEC**
C2C total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_TOTAL_RX_PER_SEC**
C2C total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_DATA_TX_PER_SEC**
C2C data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_DATA_RX_PER_SEC**
C2C data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK0_TOTAL_TX_PER_SEC**
C2C link 0 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK0_TOTAL_RX_PER_SEC**
C2C link 0 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK0_DATA_TX_PER_SEC**
C2C link 0 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK0_DATA_RX_PER_SEC**
C2C link 0 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK1_TOTAL_TX_PER_SEC**
C2C link 1 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK1_TOTAL_RX_PER_SEC**
C2C link 1 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK1_DATA_TX_PER_SEC**
C2C link 1 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK1_DATA_RX_PER_SEC**
C2C link 1 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK2_TOTAL_TX_PER_SEC**
C2C link 2 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK2_TOTAL_RX_PER_SEC**
C2C link 2 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK2_DATA_TX_PER_SEC**
C2C link 2 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK2_DATA_RX_PER_SEC**
C2C link 2 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK3_TOTAL_TX_PER_SEC**
C2C link 3 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK3_TOTAL_RX_PER_SEC**
C2C link 3 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK3_DATA_TX_PER_SEC**
C2C link 3 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK3_DATA_RX_PER_SEC**
C2C link 3 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK4_TOTAL_TX_PER_SEC**
C2C link 4 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK4_TOTAL_RX_PER_SEC**
C2C link 4 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK4_DATA_TX_PER_SEC**
C2C link 4 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK4_DATA_RX_PER_SEC**
C2C link 4 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK5_TOTAL_TX_PER_SEC**
C2C link 5 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK5_TOTAL_RX_PER_SEC**
C2C link 5 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK5_DATA_TX_PER_SEC**
C2C link 5 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK5_DATA_RX_PER_SEC**
C2C link 5 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK6_TOTAL_TX_PER_SEC**
C2C link 6 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK6_TOTAL_RX_PER_SEC**
C2C link 6 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK6_DATA_TX_PER_SEC**
C2C link 6 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK6_DATA_RX_PER_SEC**
C2C link 6 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK7_TOTAL_TX_PER_SEC**
C2C link 7 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK7_TOTAL_RX_PER_SEC**
C2C link 7 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK7_DATA_TX_PER_SEC**
C2C link 7 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK7_DATA_RX_PER_SEC**
C2C link 7 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK8_TOTAL_TX_PER_SEC**
C2C link 8 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK8_TOTAL_RX_PER_SEC**
C2C link 8 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK8_DATA_TX_PER_SEC**
C2C link 8 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK8_DATA_RX_PER_SEC**
C2C link 8 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK9_TOTAL_TX_PER_SEC**
C2C link 9 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK9_TOTAL_RX_PER_SEC**
C2C link 9 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK9_DATA_TX_PER_SEC**
C2C link 9 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK9_DATA_RX_PER_SEC**
C2C link 9 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK10_TOTAL_TX_PER_SEC**
C2C link 10 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK10_TOTAL_RX_PER_SEC**
C2C link 10 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK10_DATA_TX_PER_SEC**
C2C link 10 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK10_DATA_RX_PER_SEC**
C2C link 10 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK11_TOTAL_TX_PER_SEC**
C2C link 11 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK11_TOTAL_RX_PER_SEC**
C2C link 11 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK11_DATA_TX_PER_SEC**
C2C link 11 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK11_DATA_RX_PER_SEC**
C2C link 11 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK12_TOTAL_TX_PER_SEC**
C2C link 12 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK12_TOTAL_RX_PER_SEC**
C2C link 12 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK12_DATA_TX_PER_SEC**
C2C link 12 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK12_DATA_RX_PER_SEC**
C2C link 12 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK13_TOTAL_TX_PER_SEC**
C2C link 13 total transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK13_TOTAL_RX_PER_SEC**
C2C link 13 total receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK13_DATA_TX_PER_SEC**
C2C link 13 data transmit bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_C2C_LINK13_DATA_RX_PER_SEC**
C2C link 13 data receive bandwidth in MiB/sec.

enumerator **NVML_GPM_METRIC_HOSTMEM_CACHE_HIT**
Percentage of host memory cache hits. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_HOSTMEM_CACHE_MISS**
Percentage of host memory cache misses. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_PEERMEM_CACHE_HIT**
Percentage of peer memory cache hits. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_PEERMEM_CACHE_MISS**
Percentage of peer memory cache misses. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_DRAM_CACHE_HIT**
Percentage of DRAM cache hits. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_DRAM_CACHE_MISS**
Percentage of DRAM cache misses. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVENC_0_UTIL**
Percent utilization of NVENC 0. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVENC_1_UTIL**
Percent utilization of NVENC 1. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVENC_2_UTIL**
Percent utilization of NVENC 2. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVENC_3_UTIL**
Percent utilization of NVENC 3. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_GR0_CTXSW_CYCLES_ELAPSED**
Total context switch cycles elapsed for GR engine 0.

enumerator **NVML_GPM_METRIC_GR0_CTXSW_CYCLES_ACTIVE**
Active context switch cycles for GR engine 0.

enumerator **NVML_GPM_METRIC_GR0_CTXSW_REQUESTS**
Number of context switch requests for GR engine 0.

enumerator **NVML_GPM_METRIC_GR0_CTXSW_CYCLES_PER_REQ**
Average context switch cycles per request for GR engine 0.

enumerator **NVML_GPM_METRIC_GR0_CTXSW_ACTIVE_PCT**
Percentage of time GR engine 0 context switches were active. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_GR1_CTXSW_CYCLES_ELAPSED**
Total context switch cycles elapsed for GR engine 1.

enumerator **NVML_GPM_METRIC_GR1_CTXSW_CYCLES_ACTIVE**
Active context switch cycles for GR engine 1.

enumerator **NVML_GPM_METRIC_GR1_CTXSW_REQUESTS**
Number of context switch requests for GR engine 1.

enumerator **NVML_GPM_METRIC_GR1_CTXSW_CYCLES_PER_REQ**
Average context switch cycles per request for GR engine 1.

enumerator **NVML_GPM_METRIC_GR1_CTXSW_ACTIVE_PCT**
Percentage of time GR engine 1 context switches were active. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_GR2_CTXSW_CYCLES_ELAPSED**
Total context switch cycles elapsed for GR engine 2.

enumerator **NVML_GPM_METRIC_GR2_CTXSW_CYCLES_ACTIVE**
Active context switch cycles for GR engine 2.

enumerator **NVML_GPM_METRIC_GR2_CTXSW_REQUESTS**
Number of context switch requests for GR engine 2.

enumerator **NVML_GPM_METRIC_GR2_CTXSW_CYCLES_PER_REQ**
Average context switch cycles per request for GR engine 2.

enumerator **NVML_GPM_METRIC_GR2_CTXSW_ACTIVE_PCT**
Percentage of time GR engine 2 context switches were active. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_GR3_CTXSW_CYCLES_ELAPSED**
Total context switch cycles elapsed for GR engine 3.

enumerator **NVML_GPM_METRIC_GR3_CTXSW_CYCLES_ACTIVE**
Active context switch cycles for GR engine 3.

enumerator **NVML_GPM_METRIC_GR3_CTXSW_REQUESTS**
Number of context switch requests for GR engine 3.

enumerator **NVML_GPM_METRIC_GR3_CTXSW_CYCLES_PER_REQ**
Average context switch cycles per request for GR engine 3.

enumerator **NVML_GPM_METRIC_GR3_CTXSW_ACTIVE_PCT**
Percentage of time GR engine 3 context switches were active. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_GR4_CTXSW_CYCLES_ELAPSED**
Total context switch cycles elapsed for GR engine 4.

enumerator **NVML_GPM_METRIC_GR4_CTXSW_CYCLES_ACTIVE**
Active context switch cycles for GR engine 4.

enumerator **NVML_GPM_METRIC_GR4_CTXSW_REQUESTS**
Number of context switch requests for GR engine 4.

enumerator **NVML_GPM_METRIC_GR4_CTXSW_CYCLES_PER_REQ**
Average context switch cycles per request for GR engine 4.

enumerator **NVML_GPM_METRIC_GR4_CTXSW_ACTIVE_PCT**
Percentage of time GR engine 4 context switches were active. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_GR5_CTXSW_CYCLES_ELAPSED**
Total context switch cycles elapsed for GR engine 5.

enumerator **NVML_GPM_METRIC_GR5_CTXSW_CYCLES_ACTIVE**
Active context switch cycles for GR engine 5.

enumerator **NVML_GPM_METRIC_GR5_CTXSW_REQUESTS**
Number of context switch requests for GR engine 5.

enumerator **NVML_GPM_METRIC_GR5_CTXSW_CYCLES_PER_REQ**
Average context switch cycles per request for GR engine 5.

enumerator **NVML_GPM_METRIC_GR5_CTXSW_ACTIVE_PCT**
Percentage of time GR engine 5 context switches were active. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_GR6_CTXSW_CYCLES_ELAPSED**
Total context switch cycles elapsed for GR engine 6.

enumerator **NVML_GPM_METRIC_GR6_CTXSW_CYCLES_ACTIVE**
Active context switch cycles for GR engine 6.

enumerator **NVML_GPM_METRIC_GR6_CTXSW_REQUESTS**
Number of context switch requests for GR engine 6.

enumerator **NVML_GPM_METRIC_GR6_CTXSW_CYCLES_PER_REQ**
Average context switch cycles per request for GR engine 6.

enumerator **NVML_GPM_METRIC_GR6_CTXSW_ACTIVE_PCT**
Percentage of time GR engine 6 context switches were active. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_GR7_CTXSW_CYCLES_ELAPSED**
Total context switch cycles elapsed for GR engine 7.

enumerator **NVML_GPM_METRIC_GR7_CTXSW_CYCLES_ACTIVE**
Active context switch cycles for GR engine 7.

enumerator **NVML_GPM_METRIC_GR7_CTXSW_REQUESTS**
Number of context switch requests for GR engine 7.

enumerator **NVML_GPM_METRIC_GR7_CTXSW_CYCLES_PER_REQ**
Average context switch cycles per request for GR engine 7.

enumerator **NVML_GPM_METRIC_GR7_CTXSW_ACTIVE_PCT**
Percentage of time GR engine 7 context switches were active. 0.0 - 100.0.

enumerator **NVML_GPM_METRIC_NVLINK_L18_RX_PER_SEC**
NvLink read bandwidth for link 18 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L18_TX_PER_SEC**
NvLink write bandwidth for link 18 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L19_RX_PER_SEC**
NvLink read bandwidth for link 19 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L19_TX_PER_SEC**
NvLink write bandwidth for link 19 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L20_RX_PER_SEC**
NvLink read bandwidth for link 20 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L20_TX_PER_SEC**
NvLink write bandwidth for link 20 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L21_RX_PER_SEC**
NvLink read bandwidth for link 21 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L21_TX_PER_SEC**
NvLink write bandwidth for link 21 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L22_RX_PER_SEC**
NvLink read bandwidth for link 22 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L22_TX_PER_SEC**
NvLink write bandwidth for link 22 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L23_RX_PER_SEC**
NvLink read bandwidth for link 23 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L23_TX_PER_SEC**
NvLink write bandwidth for link 23 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L24_RX_PER_SEC**
NvLink read bandwidth for link 24 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L24_TX_PER_SEC**
NvLink write bandwidth for link 24 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L25_RX_PER_SEC**
NvLink read bandwidth for link 25 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L25_TX_PER_SEC**
NvLink write bandwidth for link 25 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L26_RX_PER_SEC**
NvLink read bandwidth for link 26 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L26_TX_PER_SEC**
NvLink write bandwidth for link 26 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L27_RX_PER_SEC**
NvLink read bandwidth for link 27 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L27_TX_PER_SEC**
NvLink write bandwidth for link 27 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L28_RX_PER_SEC**
NvLink read bandwidth for link 28 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L28_TX_PER_SEC**
NvLink write bandwidth for link 28 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L29_RX_PER_SEC**
NvLink read bandwidth for link 29 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L29_TX_PER_SEC**
NvLink write bandwidth for link 29 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L30_RX_PER_SEC**
NvLink read bandwidth for link 30 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L30_TX_PER_SEC**
NvLink write bandwidth for link 30 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L31_RX_PER_SEC**
NvLink read bandwidth for link 31 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L31_TX_PER_SEC**
NvLink write bandwidth for link 31 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L32_RX_PER_SEC**
NvLink read bandwidth for link 32 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L32_TX_PER_SEC**
NvLink write bandwidth for link 32 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L33_RX_PER_SEC**
NvLink read bandwidth for link 33 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L33_TX_PER_SEC**
NvLink write bandwidth for link 33 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L34_RX_PER_SEC**
NvLink read bandwidth for link 34 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L34_TX_PER_SEC**
NvLink write bandwidth for link 34 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L35_RX_PER_SEC**
NvLink read bandwidth for link 35 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L35_TX_PER_SEC**
NvLink write bandwidth for link 35 in MiB/sec.

enumerator **NVML_GPM_METRIC_SM_CYCLES_ELAPSED**
The GPU's SM cycles elapsed since reboot.

enumerator **NVML_GPM_METRIC_SM_CYCLES_ACTIVE**
The GPU's SM activity since reboot.

enumerator **NVML_GPM_METRIC_MMA_CYCLES_ACTIVE**
The GPU's SM MMA tensor activity since reboot.

enumerator **NVML_GPM_METRIC_DMMA_CYCLES_ACTIVE**
The GPU's SM DMMA tensor activity since reboot.

enumerator **NVML_GPM_METRIC_HMMA_CYCLES_ACTIVE**
The GPU's SM HMMA tensor activity since reboot.

enumerator **NVML_GPM_METRIC_IMMA_CYCLES_ACTIVE**
The GPU's SM IMMA tensor activity since reboot.

enumerator **NVML_GPM_METRIC_DFMA_CYCLES_ACTIVE**
The GPU's SM DFMA tensor activity since reboot.

enumerator **NVML_GPM_METRIC_PCIE_TX**
The PCIe TX traffic since reboot.

enumerator **NVML_GPM_METRIC_PCIE_RX**
The PCIe RX traffic since reboot.

enumerator **NVML_GPM_METRIC_INTEGER_CYCLES_ACTIVE**
The GPU's SM integer activity since reboot.

enumerator **NVML_GPM_METRIC_FP64_CYCLES_ACTIVE**
The GPU's SM FP64 activity since reboot.

enumerator **NVML_GPM_METRIC_FP32_CYCLES_ACTIVE**
The GPU's SM FP32 activity since reboot.

enumerator **NVML_GPM_METRIC_FP16_CYCLES_ACTIVE**
The GPU's SM FP16 activity since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L0_RX**
NvLink read for link 0 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L0_TX**
NvLink write for link 0 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L1_RX**
NvLink read for link 1 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L1_TX**
NvLink write for link 1 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L2_RX**
NvLink read for link 2 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L2_TX**
NvLink write for link 2 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L3_RX**
NvLink read for link 3 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L3_TX**
NvLink write for link 3 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L4_RX**
NvLink read for link 4 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L4_TX**
NvLink write for link 4 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L5_RX**
NvLink read for link 5 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L5_TX**
NvLink write for link 5 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L6_RX**
NvLink read for link 6 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L6_TX**
NvLink write for link 6 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L7_RX**
NvLink read for link 7 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L7_TX**
NvLink write for link 7 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L8_RX**
NvLink read for link 8 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L8_TX**
NvLink write for link 8 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L9_RX**
NvLink read for link 9 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L9_TX**
NvLink write for link 9 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L10_RX**
NvLink read for link 10 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L10_TX**
NvLink write for link 10 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L11_RX**
NvLink read for link 11 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L11_TX**
NvLink write for link 11 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L12_RX**
NvLink read for link 12 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L12_TX**
NvLink write for link 12 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L13_RX**
NvLink read for link 13 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L13_TX**
NvLink write for link 13 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L14_RX**
NvLink read for link 14 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L14_TX**
NvLink write for link 14 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L15_RX**
NvLink read for link 15 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L15_TX**
NvLink write for link 15 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L16_RX**
NvLink read for link 16 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L16_TX**
NvLink write for link 16 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L17_RX**
NvLink read for link 17 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L17_TX**
NvLink write for link 17 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L18_RX**
NvLink read for link 18 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L18_TX**
NvLink write for link 18 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L19_RX**
NvLink read for link 19 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L19_TX**
NvLink write for link 19 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L20_RX**
NvLink read for link 20 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L20_TX**
NvLink write for link 20 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L21_RX**
NvLink read for link 21 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L21_TX**
NvLink write for link 21 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L22_RX**
NvLink read for link 22 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L22_TX**
NvLink write for link 22 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L23_RX**
NvLink read for link 23 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L23_TX**
NvLink write for link 23 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L24_RX**
NvLink read for link 24 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L24_TX**
NvLink write for link 24 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L25_RX**
NvLink read for link 25 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L25_TX**
NvLink write for link 25 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L26_RX**
NvLink read for link 26 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L26_TX**
NvLink write for link 26 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L27_RX**
NvLink read for link 27 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L27_TX**
NvLink write for link 27 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L28_RX**
NvLink read for link 28 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L28_TX**
NvLink write for link 28 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L29_RX**
NvLink read for link 29 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L29_TX**
NvLink write for link 29 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L30_RX**
NvLink read for link 30 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L30_TX**
NvLink write for link 30 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L31_RX**
NvLink read for link 31 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L31_TX**
NvLink write for link 31 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L32_RX**
NvLink read for link 32 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L32_TX**
NvLink write for link 32 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L33_RX**
NvLink read for link 33 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L33_TX**
NvLink write for link 33 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L34_RX**
NvLink read for link 34 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L34_TX**
NvLink write for link 34 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L35_RX**
NvLink read for link 35 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L35_TX**
NvLink write for link 35 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L36_RX**
NvLink read for link 36 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L36_TX**
NvLink write for link 36 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L37_RX**
NvLink read for link 37 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L37_TX**
NvLink write for link 37 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L38_RX**
NvLink read for link 38 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L38_TX**
NvLink write for link 38 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L39_RX**
NvLink read for link 39 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L39_TX**
NvLink write for link 39 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L40_RX**
NvLink read for link 40 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L40_TX**
NvLink write for link 40 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L41_RX**
NvLink read for link 41 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L41_TX**
NvLink write for link 41 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L42_RX**
NvLink read for link 42 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L42_TX**
NvLink write for link 42 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L43_RX**
NvLink read for link 43 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L43_TX**
NvLink write for link 43 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L44_RX**
NvLink read for link 44 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L44_TX**
NvLink write for link 44 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L45_RX**
NvLink read for link 45 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L45_TX**
NvLink write for link 45 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L46_RX**
NvLink read for link 46 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L46_TX**
NvLink write for link 46 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L47_RX**
NvLink read for link 47 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L47_TX**
NvLink write for link 47 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L48_RX**
NvLink read for link 48 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L48_TX**
NvLink write for link 48 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L49_RX**
NvLink read for link 49 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L49_TX**
NvLink write for link 49 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L50_RX**
NvLink read for link 50 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L50_TX**
NvLink write for link 50 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L51_RX**
NvLink read for link 51 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L51_TX**
NvLink write for link 51 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L52_RX**
NvLink read for link 52 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L52_TX**
NvLink write for link 52 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L53_RX**
NvLink read for link 53 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L53_TX**
NvLink write for link 53 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L54_RX**
NvLink read for link 54 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L54_TX**
NvLink write for link 54 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L55_RX**
NvLink read for link 55 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L55_TX**
NvLink write for link 55 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L56_RX**
NvLink read for link 56 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L56_TX**
NvLink write for link 56 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L57_RX**
NvLink read for link 57 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L57_TX**
NvLink write for link 57 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L58_RX**
NvLink read for link 58 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L58_TX**
NvLink write for link 58 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L59_RX**
NvLink read for link 59 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L59_TX**
NvLink write for link 59 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L60_RX**
NvLink read for link 60 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L60_TX**
NvLink write for link 60 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L61_RX**
NvLink read for link 61 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L61_TX**
NvLink write for link 61 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L62_RX**
NvLink read for link 62 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L62_TX**
NvLink write for link 62 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L63_RX**
NvLink read for link 63 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L63_TX**
NvLink write for link 63 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L64_RX**
NvLink read for link 64 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L64_TX**
NvLink write for link 64 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L65_RX**
NvLink read for link 65 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L65_TX**
NvLink write for link 65 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L66_RX**
NvLink read for link 66 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L66_TX**
NvLink write for link 66 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L67_RX**
NvLink read for link 67 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L67_TX**
NvLink write for link 67 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L68_RX**
NvLink read for link 68 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L68_TX**
NvLink write for link 68 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L69_RX**
NvLink read for link 69 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L69_TX**
NvLink write for link 69 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L70_RX**
NvLink read for link 70 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L70_TX**
NvLink write for link 70 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L71_RX**
NvLink read for link 71 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L71_TX**
NvLink write for link 71 in bytes since reboot.

enumerator **NVML_GPM_METRIC_NVLINK_L36_RX_PER_SEC**
NvLink read bandwidth for link 36 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L36_TX_PER_SEC**
NvLink write bandwidth for link 36 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L37_RX_PER_SEC**
NvLink read bandwidth for link 37 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L37_TX_PER_SEC**
NvLink write bandwidth for link 37 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L38_RX_PER_SEC**
NvLink read bandwidth for link 38 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L38_TX_PER_SEC**
NvLink write bandwidth for link 38 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L39_RX_PER_SEC**
NvLink read bandwidth for link 39 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L39_TX_PER_SEC**
NvLink write bandwidth for link 39 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L40_RX_PER_SEC**
NvLink read bandwidth for link 40 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L40_TX_PER_SEC**
NvLink write bandwidth for link 40 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L41_RX_PER_SEC**
NvLink read bandwidth for link 41 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L41_TX_PER_SEC**
NvLink write bandwidth for link 41 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L42_RX_PER_SEC**
NvLink read bandwidth for link 42 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L42_TX_PER_SEC**
NvLink write bandwidth for link 42 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L43_RX_PER_SEC**
NvLink read bandwidth for link 43 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L43_TX_PER_SEC**
NvLink write bandwidth for link 43 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L44_RX_PER_SEC**
NvLink read bandwidth for link 44 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L44_TX_PER_SEC**
NvLink write bandwidth for link 44 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L45_RX_PER_SEC**
NvLink read bandwidth for link 45 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L45_TX_PER_SEC**
NvLink write bandwidth for link 45 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L46_RX_PER_SEC**
NvLink read bandwidth for link 46 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L46_TX_PER_SEC**
NvLink write bandwidth for link 46 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L47_RX_PER_SEC**
NvLink read bandwidth for link 47 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L47_TX_PER_SEC**
NvLink write bandwidth for link 47 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L48_RX_PER_SEC**
NvLink read bandwidth for link 48 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L48_TX_PER_SEC**
NvLink write bandwidth for link 48 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L49_RX_PER_SEC**
NvLink read bandwidth for link 49 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L49_TX_PER_SEC**
NvLink write bandwidth for link 49 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L50_RX_PER_SEC**
NvLink read bandwidth for link 50 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L50_TX_PER_SEC**
NvLink write bandwidth for link 50 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L51_RX_PER_SEC**
NvLink read bandwidth for link 51 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L51_TX_PER_SEC**
NvLink write bandwidth for link 51 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L52_RX_PER_SEC**
NvLink read bandwidth for link 52 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L52_TX_PER_SEC**
NvLink write bandwidth for link 52 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L53_RX_PER_SEC**
NvLink read bandwidth for link 53 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L53_TX_PER_SEC**
NvLink write bandwidth for link 53 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L54_RX_PER_SEC**
NvLink read bandwidth for link 54 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L54_TX_PER_SEC**
NvLink write bandwidth for link 54 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L55_RX_PER_SEC**
NvLink read bandwidth for link 55 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L55_TX_PER_SEC**
NvLink write bandwidth for link 55 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L56_RX_PER_SEC**
NvLink read bandwidth for link 56 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L56_TX_PER_SEC**
NvLink write bandwidth for link 56 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L57_RX_PER_SEC**
NvLink read bandwidth for link 57 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L57_TX_PER_SEC**
NvLink write bandwidth for link 57 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L58_RX_PER_SEC**
NvLink read bandwidth for link 58 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L58_TX_PER_SEC**
NvLink write bandwidth for link 58 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L59_RX_PER_SEC**
NvLink read bandwidth for link 59 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L59_TX_PER_SEC**
NvLink write bandwidth for link 59 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L60_RX_PER_SEC**
NvLink read bandwidth for link 60 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L60_TX_PER_SEC**
NvLink write bandwidth for link 60 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L61_RX_PER_SEC**
NvLink read bandwidth for link 61 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L61_TX_PER_SEC**
NvLink write bandwidth for link 61 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L62_RX_PER_SEC**
NvLink read bandwidth for link 62 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L62_TX_PER_SEC**
NvLink write bandwidth for link 62 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L63_RX_PER_SEC**
NvLink read bandwidth for link 63 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L63_TX_PER_SEC**
NvLink write bandwidth for link 63 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L64_RX_PER_SEC**
NvLink read bandwidth for link 64 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L64_TX_PER_SEC**
NvLink write bandwidth for link 64 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L65_RX_PER_SEC**
NvLink read bandwidth for link 65 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L65_TX_PER_SEC**
NvLink write bandwidth for link 65 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L66_RX_PER_SEC**
NvLink read bandwidth for link 66 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L66_TX_PER_SEC**
NvLink write bandwidth for link 66 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L67_RX_PER_SEC**
NvLink read bandwidth for link 67 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L67_TX_PER_SEC**
NvLink write bandwidth for link 67 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L68_RX_PER_SEC**
NvLink read bandwidth for link 68 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L68_TX_PER_SEC**
NvLink write bandwidth for link 68 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L69_RX_PER_SEC**
NvLink read bandwidth for link 69 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L69_TX_PER_SEC**
NvLink write bandwidth for link 69 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L70_RX_PER_SEC**
NvLink read bandwidth for link 70 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L70_TX_PER_SEC**
NvLink write bandwidth for link 70 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L71_RX_PER_SEC**
NvLink read bandwidth for link 71 in MiB/sec.

enumerator **NVML_GPM_METRIC_NVLINK_L71_TX_PER_SEC**
NvLink write bandwidth for link 71 in MiB/sec.

enumerator **NVML_GPM_METRIC_MAX**
Maximum value above +1.

5.2.1.22.2 GPM Functions

5.2.1.22.2.1 Functions

nvmlReturn_t *nvmlGpmMetricsGet*(nvmlGpmMetricsGet_t *metricsGet)
Calculate GPM metrics from two samples.

nvmlReturn_t *nvmlGpmMigSampleGet*(nvmlDevice_t device, unsigned int gpuInstanceId, nvmlGpmSample_t gpmSample)
Read a sample of GPM metrics into the provided *gpmSample* buffer for a MIG GPU Instance.

nvmlReturn_t *nvmlGpmQueryDeviceSupport*(nvmlDevice_t device, nvmlGpmSupport_t *gpmSupport)
Indicate whether the supplied device supports GPM.

nvmlReturn_t *nvmlGpmQueryIfStreamingEnabled*(nvmlDevice_t device, unsigned int *state)
Get GPM stream state.

nvmlReturn_t *nvmlGpmSampleAlloc*(nvmlGpmSample_t *gpmSample)
Allocate a sample buffer to be used with NVML GPM .

nvmlReturn_t *nvmlGpmSampleFree*(nvmlGpmSample_t gpmSample)
Free an allocated sample buffer that was allocated with *nvmlGpmSampleAlloc()*

nvmlReturn_t *nvmlGpmSampleGet*(nvmlDevice_t device, nvmlGpmSample_t gpmSample)
Read a sample of GPM metrics into the provided *gpmSample* buffer.

nvmlReturn_t *nvmlGpmSetStreamingEnabled*(nvmlDevice_t device, unsigned int state)
Set GPM stream state.

5.2.1.22.2.2 Functions

nvmlReturn_t **nvmlGpmMetricsGet**(*nvmlGpmMetricsGet_t* *metricsGet)
Calculate GPM metrics from two samples.

For Hopper or newer fully supported devices.

To retrieve metrics, the user must first allocate the two sample buffers at *metricsGet->sample1* and *metricsGet->sample2* by calling nvmlGpmSampleAlloc(). Next, the user should fill in the ID of each metric in *metricsGet->metrics*[i].metricId and specify the total number of metrics to retrieve

in *metricsGet->numMetrics*, The version should be set to NVML_GPM_METRICS_GET_VERSION in *metricsGet->version*. The user then calls the nvmlGpmSampleGet() API twice to obtain 2 samples of counters.

Note

The interval between these two nvmlGpmSampleGet() calls should be greater than 100ms due to the internal sample refresh rate. Finally, the user calls *nvmlGpmMetricsGet* to retrieve the metrics, which will be stored at *metricsGet->metrics*

Parameters

metricsGet – IN/OUT: populated nvmlGpmMetricsGet_t struct

Returns

- *NVML_SUCCESS* on success
- Nonzero NVML_ERROR_? enum on error

nvmlReturn_t **nvmlGpmMigSampleGet**(
nvmlDevice_t device, unsigned int gpuInstanceId, *nvmlGpmSample_t* gpmSample,
)

Read a sample of GPM metrics into the provided *gpmSample* buffer for a MIG GPU Instance.

After two samples are gathered, you can call nvmlGpmMetricGet on those samples to retrive metrics

For Hopper or newer fully supported devices.

Note

The interval between two nvmlGpmMigSampleGet() calls should be greater than 100ms due to the internal sample refresh rate.

Parameters

- **device** – Device to get samples for
- **gpuInstanceId** – MIG GPU Instance ID
- **gpmSample** – Buffer to read samples into

Returns

- *NVML_SUCCESS* on success
- Nonzero NVML_ERROR_? enum on error

nvmlReturn_t **nvmlGpmQueryDeviceSupport**(
nvmlDevice_t device, *nvmlGpmSupport_t* *gpmSupport,
)

Indicate whether the supplied device supports GPM.

For Hopper or newer fully supported devices.

Note

This API supports the device handle and MIG device handle.

Parameters

- **device** – The device handle or MIG device handle to query for
- **gpmSupport** – Structure to indicate GPM support nvmlGpmSupport_t. Indicates GPM support per system for the supplied device

Returns

- NVML_SUCCESS on success
- Nonzero NVML_ERROR_? enum if there is an error in processing the query

nvmlReturn_t **nvmlGpmQueryIfStreamingEnabled**(
nvmlDevice_t device, unsigned int *state,
)

Get GPM stream state.

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

Parameters

- **device** – The identifier of the target device
- **state** – Returns GPM stream state NVML_FEATURE_DISABLED or NVML_FEATURE_ENABLED

Returns

- *NVML_SUCCESS* if *current* GPM stream state were successfully queried
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *state* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device

nvmlReturn_t **nvmlGpmSampleAlloc**(*nvmlGpmSample_t* *gpmSample)

Allocate a sample buffer to be used with NVML GPM .

You will need to allocate at least two of these buffers to use with the NVML GPM feature

For Hopper or newer fully supported devices.

Parameters

gpmSample – Where the allocated sample will be stored

Returns

- *NVML_SUCCESS* on success
- *NVML_ERROR_INVALID_ARGUMENT* if an invalid pointer is provided
- *NVML_ERROR_MEMORY* if system memory is insufficient

nvmlReturn_t **nvmlGpmSampleFree**(*nvmlGpmSample_t* gpmSample)

Free an allocated sample buffer that was allocated with *nvmlGpmSampleAlloc()*

For Hopper or newer fully supported devices.

Parameters

gpmSample – Sample to free

Returns

- *NVML_SUCCESS* on success
- *NVML_ERROR_INVALID_ARGUMENT* if an invalid pointer is provided

nvmlReturn_t **nvmlGpmSampleGet**(
nvmlDevice_t device, *nvmlGpmSample_t* gpmSample,
)

Read a sample of GPM metrics into the provided *gpmSample* buffer.

After two samples are gathered, you can call nvmlGpmMetricGet on those samples to retrive metrics

For Hopper or newer fully supported devices.

Note

The interval between two nvmlGpmSampleGet() calls should be greater than 100ms due to the internal sample refresh rate.

Note

This API supports the device handle and MIG device handle.

Parameters

- **device** – The device handle or MIG device handle to get samples for
- **gpmSample** – Buffer to read samples into

Returns

- *NVML_SUCCESS* on success
- Nonzero NVML_ERROR_? enum on error

nvmlReturn_t **nvmlGpmSetStreamingEnabled**(
nvmlDevice_t device, unsigned int state,
)

Set GPM stream state.

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

Parameters

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

- **state** – GPM stream state, NVML_FEATURE_DISABLED or NVML_FEATURE_ENABLED

Returns

- *NVML_SUCCESS* if *current* GPM stream state is successfully set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device

5.2.1.22.3 GPM Structs

5.2.1.22.3.1 Macros

NVML_GPM_METRICS_GET_VERSION

NVML_GPM_SUPPORT_VERSION

5.2.1.22.3.2 Structs

nvmlGpmMetric_t
GPM metric information.

nvmlGpmMetricsGet_t
GPM buffer information.

nvmlGpmSupport_t
GPM device information.

5.2.1.22.3.3 Typedefs

nvmlGpmSample_t
Handle to an allocated GPM sample allocated with *nvmlGpmSampleAlloc()* .

5.2.1.22.3.4 Macros

NVML_GPM_METRICS_GET_VERSION

NVML_GPM_SUPPORT_VERSION

5.2.1.22.3.5 Typedefs

typedef struct nvmlGpmSample_st ***nvmlGpmSample_t**

Handle to an allocated GPM sample allocated with *nvmlGpmSampleAlloc()*.

Free this with *nvmlGpmSampleFree()*.

5.2.1.22.4 Groups

GPM Enums

GPM Functions

GPM Structs

5.2.1.23 NvLink Methods

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

5.2.1.23.1 Macros

NVML_NVLINK_BER_EXP_SHIFT
Shift for NVLink BER exponent.

NVML_NVLINK_BER_EXP_WIDTH
Width for NVLink BER exponent.

NVML_NVLINK_BER_MANTISSA_SHIFT
Shift for NVLink BER mantissa.

NVML_NVLINK_BER_MANTISSA_WIDTH
Width for NVLink BER mantissa.

NVML_NVLINK_ERROR_COUNTER_BER_GET
Nvlink Error counter BER can be obtained using the below macros Ex - *NVML_NVLINK_ERROR_COUNTER_BER_GET(var, BER_MANTISSA)*

NVML_NVLINK_FIRMWARE_UCODE_TYPE_MSE
MSE ucode type.

NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR
NETIR ucode type.

NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_CLN
NETIR CLN ucode type.

NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_DLN
NETIR DLN ucode type.

NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_UPHY
NETIR UPHY ucode type.

NVML_NVLINK_FIRMWARE_VERSION_LENGTH
Length of firmware version string.

NVML_NVLINK_MAX_REMAP_TABLE_ENTRIES_V2
Maximum number of entries in the NVLink FLA/SPA(GPA) remap tables.

NVML_NVLINK_NVLE_AES_256_GCM_KEY_SIZE_BYTES
Size in bytes of the AES-256-GCM key secret used for NVLink encryption.

NVML_NVLINK_NVLE_PRIVATE_INFO_SIZE
Size in bytes of the NVLink NVLE private (opaque) info buffer.

NVML_NVLINK_REMAP_TABLE_ENTRIES_CHUNK

NVML_NVLINK_STATE_ACTIVE
NVLink is active.

NVML_NVLINK_STATE_ACTIVE_TRAFFIC_DISABLED
NVLink is active, but not usable for traffic.

NVML_NVLINK_STATE_INACTIVE
NVLink is inactive.

NVML_NVLINK_STATE_SLEEP
NVLink is in sleep state.

NVML_NVLINK_TOTAL_SUPPORTED_BW_MODES
Total supported NVLink bandwidth modes.

nvmlNvLinkInfo_v1
Version macro for *nvmlNvLinkInfo_v1_t* .

nvmlNvLinkInfo_v2
Version macro for *nvmlNvLinkInfo_v2_t* .

nvmlNvlinkGetBwMode_v1
Version macro for *nvmlNvlinkGetBwMode_v1_t* .

nvmlNvlinkSetBwMode_v1
Version macro for *nvmlNvlinkSetBwMode_v1_t* .

nvmlNvlinkSupportedBwModes_v1
Version macro for *nvmlNvlinkSupportedBwModes_v1_t* .

5.2.1.23.2 Enumerations

nvmlNvlinkTelemetrySampleType_t
Per-link NVLink telemetry sample types.

nvmlNvlinkVersion_t
Represents Nvlink Version.

5.2.1.23.3 Functions

nvmlReturn_t *nvmlDeviceFreezeNvLinkUtilizationCounter*(nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlEnableState_t freeze)

nvmlReturn_t *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.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceGetNvLinkInfo*(nvmlDevice_t device, nvmlNvLinkInfo_t *info)
Query NVLINK information associated with this device.

nvmlReturn_t *nvmlDeviceGetNvLinkInfo_v3*(nvmlDevice_t device, nvmlNvLinkInfo_v3_t *info)
Query NVLINK information associated with this device (v3).

nvmlReturn_t *nvmlDeviceGetNvLinkRemoteDeviceType*(nvmlDevice_t device, unsigned int link, nvmlIntNvLinkDeviceType_t *pNvLinkDeviceType)
Get the NVLink device type of the remote device connected over the given link.

nvmlReturn_t *nvmlDeviceGetNvLinkRemotePciInfo_v2*(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.

nvmlReturn_t *nvmlDeviceGetNvLinkState*(nvmlDevice_t device, unsigned int link, nvmlEnableState_t *isActive)
Retrieves the state of the device's NvLink for the link specified.

nvmlReturn_t *nvmlDeviceGetNvLinkTelemetrySamples_v1*(nvmlDevice_t device, nvmlNvlinkTelemetrySamples_v1_t *samples)
Retrieve a batch of historical NVLink per-link telemetry samples.

nvmlReturn_t *nvmlDeviceGetNvLinkUtilizationControl*(nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlNvLinkUtilizationControl_t *control)

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

nvmlReturn_t *nvmlDeviceGetNvLinkVersion*(nvmlDevice_t device, unsigned int link, unsigned int *version)
Retrieves the version of the device's NvLink for the link specified.

nvmlReturn_t *nvmlDeviceGetNvlinkBwMode*(nvmlDevice_t device, nvmlNvlinkGetBwMode_t *getBwMode)
Get the NvLink Reduced Bandwidth Mode for the device.

nvmlReturn_t *nvmlDeviceGetNvlinkNvlePrivateInfo_v1*(nvmlDevice_t device, nvmlNvlinkNvlePrivateInfo_v1_t *privateInfo)
Get NVLink NVLE private (opaque) information.

nvmlReturn_t *nvmlDeviceGetNvlinkRemapTableInfo_v1*(nvmlDevice_t device, nvmlNvlinkGetRemapTableInfo_v1_t *remapTableInfo)
Get the NVLink remap table information.

nvmlReturn_t *nvmlDeviceGetNvlinkSupportedBwModes*(nvmlDevice_t device, nvmlNvlinkSupportedBwModes_t *supportedBwMode)
Get the supported NvLink Reduced Bandwidth Modes of the device.

nvmlReturn_t *nvmlDeviceLockNvlinkRemapTable_v1*(nvmlDevice_t device, nvmlNvlinkLockRemapTable_v1_t *remapTableLockInfo)
Lock the NVLink remap table to prevent MSE from making further updates.

nvmlReturn_t *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.

nvmlReturn_t *nvmlDeviceResetNvLinkUtilizationCounter*(nvmlDevice_t device, unsigned int link, unsigned int counter)

nvmlReturn_t *nvmlDeviceSetNvLinkDeviceLowPowerThreshold*(nvmlDevice_t device, nvmlNvLinkPowerThres_t *info)
Set NvLink Low Power Threshold for device.

nvmlReturn_t *nvmlDeviceSetNvLinkUtilizationControl*(nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlNvLinkUtilizationControl_t *control, unsigned int reset)

nvmlReturn_t *nvmlDeviceSetNvlinkBwMode*(nvmlDevice_t device, nvmlNvlinkSetBwMode_t *setBwMode)
Set the NvLink Reduced Bandwidth Mode for the device.

nvmlReturn_t *nvmlDeviceSetNvlinkBwModeAsync_v1*(nvmlDevice_t device, nvmlNvlinkSetBwModeAsync_v1_t *setBwModeAsync)
Set the NvLink Reduced Bandwidth Mode asynchronously for the device.

nvmlReturn_t *nvmlDeviceSetNvlinkNvleReady_v1*(nvmlDevice_t device, nvmlNvlinkSetNvleReady_v1_t *setNvleReady)
Set the NVLink NVLE ready state.

nvmlReturn_t *nvmlDeviceSetupNvlinkNvleEncryptionKey_v1*(nvmlDevice_t device, nvmlNvlinkSetupNvleEncryptionKey_v1_t *encryptionKey)
Setup the NVLink NVLE encryption key.

nvmlReturn_t *nvmlSystemGetNvlinkBwMode*(unsigned int *nvlinkBwMode)
Get the global nvlink bandwith mode.

nvmlReturn_t *nvmlSystemSetNvlinkBwMode*(unsigned int nvlinkBwMode)
Set the global nvlink bandwith mode.

5.2.1.23.4 Structs

nvmlNvLinkInfo_v1_t
Struct to represent per device NVLINK information v1.

nvmlNvLinkInfo_v2_t
Struct to represent per device NVLINK information v2.

nvmlNvLinkInfo_v3_t
Struct to represent per device NVLINK information v3.

nvmlNvlinkFirmwareInfo_t
Struct to represent NVLINK firmware information.

nvmlNvlinkFirmwareVersion_t
Struct to represent NVLINK firmware Semantic versioning and ucode type.

nvmlNvlinkGetBwMode_v1_t

nvmlNvlinkGetRemapTableInfo_v1_t
Structure for NVLink remap table information.

nvmlNvlinkLockRemapTable_v1_t
Structure for the NVLink remap table and MSE lock operation.

nvmlNvlinkNvlePrivateInfo_v1_t
Structure for NVLink NVLE private (opaque) information.

nvmlNvlinkSetBwModeAsync_v1_t
Describes the parameters involved to setting Nvlink RBM mode asynchronously.

nvmlNvlinkSetBwMode_v1_t

nvmlNvlinkSetNvleReady_v1_t
Structure for the NVLink NVLE ready state.

nvmlNvlinkSetupNvleEncryptionKey_v1_t
Structure for NVLink NVLE encryption key setup.

nvmlNvlinkSupportedBwModes_v1_t

nvmlNvlinkTelemetrySample_v1_t
Struct representing one (link, metric) telemetry request / response slot.

nvmlNvlinkTelemetrySamples_v1_t
Batched NVLink telemetry request.

5.2.1.23.5 Typedefs

nvmlNvLinkInfo_t

nvmlNvlinkGetBwMode_t

nvmlNvlinkSetBwMode_t

nvmlNvlinkSupportedBwModes_t

5.2.1.23.6 Macros

NVML_NVLINK_BER_EXP_SHIFT
Shift for NVLink BER exponent.

NVML_NVLINK_BER_EXP_WIDTH
Width for NVLink BER exponent.

NVML_NVLINK_BER_MANTISSA_SHIFT
Shift for NVLink BER mantissa.

NVML_NVLINK_BER_MANTISSA_WIDTH
Width for NVLink BER mantissa.

NVML_NVLINK_ERROR_COUNTER_BER_GET(var, type)
Nvlink Error counter BER can be obtained using the below macros Ex - *NVML_NVLINK_ERROR_COUNTER_BER_GET(var, BER_MANTISSA)*

NVML_NVLINK_FIRMWARE_UCODE_TYPE_MSE
MSE ucode type.

NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR
NETIR ucode type.

NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_CLN
NETIR CLN ucode type.

NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_DLN
NETIR DLN ucode type.

NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_UPHY
NETIR UPHY ucode type.

NVML_NVLINK_FIRMWARE_VERSION_LENGTH
Length of firmware version string.

NVML_NVLINK_MAX_REMAP_TABLE_ENTRIES_V2
Maximum number of entries in the NVLink FLA/SPA(GPA) remap tables.

NVML_NVLINK_NVLE_AES_256_GCM_KEY_SIZE_BYTES
Size in bytes of the AES-256-GCM key secret used for NVLink encryption.

NVML_NVLINK_NVLE_PRIVATE_INFO_SIZE
Size in bytes of the NVLink NVLE private (opaque) info buffer.

NVML_NVLINK_REMAP_TABLE_ENTRIES_CHUNK

NVML_NVLINK_STATE_ACTIVE
NVLink is active.

NVML_NVLINK_STATE_ACTIVE_TRAFFIC_DISABLED
NVLink is active, but not usable for traffic.

NVML_NVLINK_STATE_INACTIVE
NVLink is inactive.

NVML_NVLINK_STATE_SLEEP
NVLink is in sleep state.

NVML_NVLINK_TOTAL_SUPPORTED_BW_MODES
Total supported NVLink bandwidth modes.

nvmlNvLinkInfo_v1
Version macro for nvmlNvLinkInfo_v1_t.

nvmlNvLinkInfo_v2
Version macro for nvmlNvLinkInfo_v2_t.

nvmlNvlinkGetBwMode_v1
Version macro for *nvmlNvlinkGetBwMode_v1_t*.

nvmlNvlinkSetBwMode_v1

Version macro for *nvmlNvlinkSetBwMode_v1_t*.

nvmlNvlinkSupportedBwModes_v1

Version macro for *nvmlNvlinkSupportedBwModes_v1_t*.

5.2.1.23.7 Enumerations

enum **nvmlNvlinkTelemetrySampleType_t**

Per-link NVLink telemetry sample types.

Values:

enumerator **NVML_NVLINK_TELEMETRY_SAMPLE_TYPE_THROUGHPUT_RAW_TX**

Raw TX flit counter for a single link.

enumerator **NVML_NVLINK_TELEMETRY_SAMPLE_TYPE_THROUGHPUT_RAW_RX**

Raw RX flit counter for a single link.

enumerator **NVML_NVLINK_TELEMETRY_SAMPLE_TYPE_COUNT**

Number of valid sample types.

enum **nvmlNvlinkVersion_t**

Represents Nvlink Version.

Values:

enumerator **NVML_NVLINK_VERSION_INVALID**

NVLink version is invalid.

enumerator **NVML_NVLINK_VERSION_1_0**

NVLink Version 1.0.

enumerator **NVML_NVLINK_VERSION_2_0**

NVLink Version 2.0.

enumerator **NVML_NVLINK_VERSION_2_2**

NVLink Version 2.2.

enumerator **NVML_NVLINK_VERSION_3_0**

NVLink Version 3.0.

enumerator **NVML_NVLINK_VERSION_3_1**

NVLink Version 3.1.

enumerator **NVML_NVLINK_VERSION_4_0**

NVLink Version 4.0.

enumerator **NVML_NVLINK_VERSION_5_0**

NVLink Version 5.0.

enumerator **NVML_NVLINK_VERSION_6_0**

NVLink Version 6.0.

5.2.1.23.8 Functions

nvmlReturn_t **nvmlDeviceFreezeNvLinkUtilizationCounter**(
nvmlDevice_t device, unsigned int link, unsigned int counter, *nvmlEnableState_t* freeze,
)

Deprecated:

Freezing NVLINK utilization counters is no longer supported.

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

For Pascal or newer fully supported devices.

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

nvmlReturn_t **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 Pascal or newer 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

nvmlReturn_t **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 Pascal or newer 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

nvmlReturn_t **nvmlDeviceGetNvLinkInfo**(
nvmlDevice_t device, *nvmlNvLinkInfo_t* *info,
)

Query NVLINK information associated with this device.

Parameters

- **device** – **[in]** The identifier of the target device
- **info** – **[out]** Reference to *nvmlNvLinkInfo_t*

Returns

- *NVML_SUCCESS* if query is success
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, or *info* is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the version is invalid/unsupported
- *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

nvmlReturn_t **nvmlDeviceGetNvLinkInfo_v3**(
    nvmlDevice_t device, *nvmlNvLinkInfo_v3_t* *info,
)

Query NVLINK information associated with this device (v3).

Parameters

- **device** – **[in]** The identifier of the target device
- **info** – **[out]** Reference to nvmlNvLinkInfo_v3_t

Returns

- *NVML_SUCCESS* if query is success
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, or *info* 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

nvmlReturn_t **nvmlDeviceGetNvLinkRemoteDeviceType**(
    nvmlDevice_t device, unsigned int link, *nvmlIntNvLinkDeviceType_t* *pNvLinkDeviceType,
)

Get the NVLink device type of the remote device connected over the given link.

Parameters

- **device** – The device handle of the target GPU
- **link** – The NVLink link index on the target GPU
- **pNvLinkDeviceType** – Pointer in which the output remote device type is returned

Returns

- *NVML_SUCCESS* if *pNvLinkDeviceType* has been set

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

nvmlReturn_t **nvmlDeviceGetNvLinkRemotePciInfo_v2**(
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 Pascal or newer 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

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

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

For Pascal or newer 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

nvmlReturn_t **nvmlDeviceGetNvLinkTelemetrySamples_v1**(
nvmlDevice_t device, *nvmlNvlinkTelemetrySamples_v1_t* *samples,
)

Retrieve a batch of historical NVLink per-link telemetry samples.

Samples are taken at approximately 100 ms intervals. Intended for use with periodic polling every ~2 seconds. Longer polling intervals are possible, but can result in dropped samples if the supported `sampleCount` is too low for the given polling interval.

RUBIN_OR_NEWER%

Parameters

- **device** – **[in]** The device handle of the GPU to retrieve samples for
- **samples** – **[inout]** Request/response batch (see *nvmlNvlinkTelemetrySamples_v1_t*)

Returns

- *NVML_SUCCESS* If the call succeeded.
- *NVML_ERROR_INVALID_ARGUMENT* If any required pointer is NULL, a given enum value is out of range, a slot's `linkId` is out of range, a slot's `sampleCount` is non-zero with a NULL `samples` pointer, or a slot's `sampleCount` is greater than the supported `sampleCount` for the given link.
- *NVML_ERROR_GPU_IS_LOST* If the target GPU has fallen off the bus or is otherwise inaccessible.
- *NVML_ERROR_NOT_SUPPORTED* If the given device does not support this API.
- *NVML_ERROR_UNKNOWN* On any unexpected error.

nvmlReturn_t **nvmlDeviceGetNvLinkUtilizationControl**(
nvmlDevice_t device, unsigned int link, unsigned int counter, *nvmlNvLinkUtilizationControl_t* *control,
)

Deprecated:

Getting utilization counter control is no longer supported.

Get the NVLINK utilization counter control information for the specified counter, 0 or 1. Please refer to nvmlNvLinkUtilizationControl_t for the structure definition

For Pascal or newer fully supported devices.

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

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

Deprecated:

Use *nvmlDeviceGetFieldValues* with NVML_FI_DEV_NVLINK_THROUGHPUT_* as field values instead.

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

For Pascal or newer fully supported devices.

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

nvmlReturn_t **nvmlDeviceGetNvLinkVersion**(
nvmlDevice_t device, unsigned int link, unsigned int *version,
)

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

For Pascal or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **link** – Specifies the NvLink link to be queried
- **version** – Requested NvLink version from *nvmlNvlinkVersion_t*

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

nvmlReturn_t **nvmlDeviceGetNvlinkBwMode**(
nvmlDevice_t device, *nvmlNvlinkGetBwMode_t* *getBwMode,
)

Get the NvLink Reduced Bandwidth Mode for the device.

For Blackwell or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **getBwMode** – Reference to *nvmlNvlinkGetBwMode_t*

Returns

- *NVML_SUCCESS* if the query was successful
- *NVML_ERROR_INVALID_ARGUMENT* if device is invalid or getBwMode is NULL
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by the device
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the version specified is not supported

nvmlReturn_t **nvmlDeviceGetNvlinkNvlePrivateInfo_v1**(
nvmlDevice_t device, *nvmlNvlinkNvlePrivateInfo_v1_t* *privateInfo,
)

Get NVLink NVLE private (opaque) information.

This API retrieves opaque NVLE private information for the given device and copies it into the *data* buffer, setting *size* to the number of bytes written.

Parameters

- **device** – **[in]** The identifier of the target device.
- **privateInfo** – **[out]** Reference to *nvmlNvlinkNvlePrivateInfo_v1_t* struct.

Returns

- *NVML_SUCCESS* if the operation completed successfully
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *privateInfo* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_UNKNOWN* if an error was encountered during the operation

nvmlReturn_t **nvmlDeviceGetNvlinkRemapTableInfo_v1**(
    nvmlDevice_t device, *nvmlNvlinkGetRemapTableInfo_v1_t* *remapTableInfo,
)

Get the NVLink remap table information.

This API returns the remap table addresses for all the entries in the FLA and SPA/GPA remap tables.

Parameters

- **device** – **[in]** The identifier of the target device.
- **remapTableInfo** – **[out]** Reference to *nvmlNvlinkGetRemapTableInfo_v1_t* struct.

Returns

- *NVML_SUCCESS* if the operation completed successfully
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *remapTableInfo* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_UNKNOWN* if an error was encountered during the operation

nvmlReturn_t **nvmlDeviceGetNvlinkSupportedBwModes**(
    nvmlDevice_t device, *nvmlNvlinkSupportedBwModes_t* *supportedBwMode,
)

Get the supported NvLink Reduced Bandwidth Modes of the device.

For Blackwell or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **supportedBwMode** – Reference to *nvmlNvlinkSupportedBwModes_t*

Returns

- *NVML_SUCCESS* if the query was successful
- *NVML_ERROR_INVALID_ARGUMENT* if device is invalid or supportedBwMode is NULL
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by the device

- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the version specified is not supported

nvmlReturn_t **nvmlDeviceLockNvlinkRemapTable_v1**(
nvmlDevice_t device, *nvmlNvlinkLockRemapTable_v1_t* *remapTableLockInfo,
)

Lock the NVLink remap table to prevent MSE from making further updates.

This API requests to lock the remap table to prevent MSE (Multi-Socket Engine) from making any further updates to the remap table for the given links.

Parameters

- **device** – **[in]** The identifier of the target device.
- **remapTableLockInfo** – **[inout]** Reference to *nvmlNvlinkLockRemapTable_v1_t* struct.

Returns

- *NVML_SUCCESS* if the operation completed successfully
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *remapTableLockInfo* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_UNKNOWN* if an error was encountered during the operation

nvmlReturn_t **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 Pascal or newer 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

nvmlReturn_t **nvmlDeviceResetNvLinkUtilizationCounter**(
nvmlDevice_t device, unsigned int link, unsigned int counter,
)

Deprecated:

Resetting NVLINK utilization counters is no longer supported.

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

For Pascal or newer fully supported devices.

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

nvmlReturn_t **nvmlDeviceSetNvLinkDeviceLowPowerThreshold**(
nvmlDevice_t device, *nvmlNvLinkPowerThres_t* *info,
)

Set NvLink Low Power Threshold for device.

For Hopper or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **info** – Reference to *nvmlNvLinkPowerThres_t* struct input parameters

Returns

- *NVML_SUCCESS* if the *Threshold* is successfully set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *Threshold* is not within range
- *NVML_ERROR_NOT_READY* if an internal driver setting prevents the threshold from being used
- *NVML_ERROR_NOT_SUPPORTED* if this query is not supported by the device

nvmlReturn_t **nvmlDeviceSetNvLinkUtilizationControl**(
nvmlDevice_t device, unsigned int link, unsigned int counter, *nvmlNvLinkUtilizationControl_t* *control, unsigned int reset,
)

Deprecated:

Setting utilization counter control is no longer supported.

Set the NVLINK utilization counter control information for the specified counter, 0 or 1. Please refer to nvmlNvLinkUtilizationControl_t for the structure definition. Performs a reset of the counters if the reset parameter is non-zero.

For Pascal or newer fully supported devices.

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

nvmlReturn_t **nvmlDeviceSetNvlinkBwMode**(
    nvmlDevice_t device, *nvmlNvlinkSetBwMode_t* *setBwMode,
)

Set the NvLink Reduced Bandwidth Mode for the device.

For Blackwell or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **setBwMode** – Reference to *nvmlNvlinkSetBwMode_t*

Returns

- *NVML_SUCCESS* if the Bandwidth mode was successfully set
- *NVML_ERROR_INVALID_ARGUMENT* if device is invalid or setBwMode is NULL
- *NVML_ERROR_NO_PERMISSION* if user does not have permission to change Bandwidth mode
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by the device
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the version specified is not supported

nvmlReturn_t **nvmlDeviceSetNvlinkBwModeAsync_v1**(
    nvmlDevice_t device, *nvmlNvlinkSetBwModeAsync_v1_t* *setBwModeAsync,
)

Set the NvLink Reduced Bandwidth Mode asynchronously for the device.

Polling should be done by checking for *NVML_GPU_FABRIC_STATE_COMPLETED* from *nvmlDeviceGetGpuFabricInfoV()*.

RUBIN_OR_NEWER%

See also

nvmlDeviceGetGpuFabricInfoV()

Parameters

- **device** – **[in]** The identifier of the target device
- **setBwModeAsync** – **[inout]** Reference to *nvmlNvlinkSetBwModeAsync_v1_t*

Returns

- *NVML_SUCCESS* if the Bandwidth mode was successfully set
- *NVML_ERROR_INVALID_ARGUMENT* if device or `setBwModeAsync` is invalid
- *NVML_ERROR_NO_PERMISSION* if user does not have permission to change Bandwidth mode
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by the device

nvmlReturn_t **nvmlDeviceSetNvlinkNvleReady_v1**(
    nvmlDevice_t device, *nvmlNvlinkSetNvleReady_v1_t* *setNvleReady,
)

Set the NVLink NVLE ready state.

This API signals whether NVLE (NVLink Encryption) is ready on the given device.

Parameters

- **device** – **[in]** The identifier of the target device.
- **setNvleReady** – **[in]** Reference to *nvmlNvlinkSetNvleReady_v1_t* struct.

Returns

- *NVML_SUCCESS* if the operation completed successfully
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *setNvleReady* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_UNKNOWN* if an error was encountered during the operation

nvmlReturn_t **nvmlDeviceSetupNvlinkNvleEncryptionKey_v1**(
    nvmlDevice_t device, *nvmlNvlinkSetupNvleEncryptionKey_v1_t* *encryptionKey,
)

Setup the NVLink NVLE encryption key.

This API sets the NVLE encryption key between GPUs with the given ALIDs.

Parameters

- **device** – **[in]** The identifier of the target device.
- **encryptionKey** – **[in]** Reference to *nvmlNvlinkSetupNvleEncryptionKey_v1_t* struct.

Returns

- *NVML_SUCCESS* if the operation completed successfully
- *NVML_ERROR_INVALID_ARGUMENT* if *device* or *encryptionKey* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if the device does not support this feature
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_UNKNOWN* if an error was encountered during the operation

nvmlReturn_t **nvmlSystemGetNvlinkBwMode**(unsigned int *nvlinkBwMode)

Get the global nvlink bandwith mode.

Parameters

nvlinkBwMode – reference of nvlink bandwidth mode

Returns

- *NVML_SUCCESS* on success
- *NVML_ERROR_INVALID_ARGUMENT* if an invalid pointer is provided
- *NVML_ERROR_NOT_SUPPORTED* if GPU is not Hopper or newer architecture.
- *NVML_ERROR_NO_PERMISSION* if not root user

nvmlReturn_t **nvmlSystemSetNvlinkBwMode**(unsigned int nvlinkBwMode)

Set the global nvlink bandwith mode.

Parameters

nvlinkBwMode – nvlink bandwidth mode

Returns

- *NVML_SUCCESS* on success
- *NVML_ERROR_INVALID_ARGUMENT* if an invalid argument is provided
- *NVML_ERROR_IN_USE* if P2P object exists
- *NVML_ERROR_NOT_SUPPORTED* if GPU is not Hopper or newer architecture.
- *NVML_ERROR_NO_PERMISSION* if not root user

5.2.1.23.9 Typedefs

typedef *nvmlNvLinkInfo_v2_t* **nvmlNvLinkInfo_t**

typedef *nvmlNvlinkGetBwMode_v1_t* **nvmlNvlinkGetBwMode_t**

typedef *nvmlNvlinkSetBwMode_v1_t* **nvmlNvlinkSetBwMode_t**

typedef *nvmlNvlinkSupportedBwModes_v1_t* **nvmlNvlinkSupportedBwModes_t**

5.2.1.24 NvmlClocksEventReasons

5.2.1.24.1 Macros

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

nvmlClocksEventReasonApplicationsClocksSetting
Clocks are set to application-specific values.

nvmlClocksEventReasonBoardLimit
Board limit.

nvmlClocksEventReasonDisplayClockSetting
GPU clocks are limited by current setting of Display clocks.

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

nvmlClocksEventReasonNone
Bit mask representing no clocks throttling.

nvmlClocksEventReasonReliability
Reliability.

nvmlClocksEventReasonSwPowerCap
The clocks have been optimized to ensure not to exceed currently set power limits.

nvmlClocksEventReasonSwThermalSlowdown
SW Thermal Slowdown.

nvmlClocksEventReasonSyncBoost
Sync Boost.

nvmlClocksThrottleReasonAll
Deprecated: All clock throttling reasons.

nvmlClocksThrottleReasonApplicationsClocksSetting
Deprecated: Application clocks setting.

nvmlClocksThrottleReasonDisplayClockSetting
Deprecated: Display clock setting.

nvmlClocksThrottleReasonGpuIdle
Deprecated: GPU idle.

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

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

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

nvmlClocksThrottleReasonNone
Deprecated: No clock throttling.

nvmlClocksThrottleReasonSwPowerCap
Deprecated: Software power cap.

nvmlClocksThrottleReasonSwThermalSlowdown
Deprecated: Software thermal slowdown.

nvmlClocksThrottleReasonSyncBoost
Deprecated: Sync boost.

nvmlClocksThrottleReasonUserDefinedClocks
Deprecated: User-defined clocks.

5.2.1.24.2 Macros

nvmlClocksEventReasonAll

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

Bitmask of all clock event reasons.

nvmlClocksEventReasonApplicationsClocksSetting

Clocks are set to application-specific values.

nvmlClocksEventReasonBoardLimit

Board limit.

The board limit (operating) policy is currently limiting the GPU clocks. Board limit policy is limiting clocks.

nvmlClocksEventReasonDisplayClockSetting

GPU clocks are limited by current setting of Display clocks.

See also

bug 1997531 Display clock setting limited.

nvmlClocksEventReasonGpuIdle

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

Note

This limiter may be removed in a later release GPU is idle.

nvmlClocksEventReasonNone

Bit mask representing no clocks throttling.

Clocks are as high as possible. No clock throttling.

nvmlClocksEventReasonReliability

Reliability.

The reliability policy is currently limiting the GPU clocks. Reliability policy is limiting clocks.

nvmlClocksEventReasonSwPowerCap

The clocks have been optimized to ensure not to exceed currently set power limits.

See also

nvmlDeviceGetPowerUsage

See also

nvmlDeviceSetPowerManagementLimit

See also

nvmlDeviceGetPowerManagementLimit Software power cap activated.

nvmlClocksEventReasonSwThermalSlowdown

SW Thermal Slowdown.

The current clocks have been optimized to ensure the the following is true:

- Current GPU temperature does not exceed GPU Max Operating Temperature
- Current memory temperature does not exceeed Memory Max Operating Temperature Software thermal slowdown activated.

nvmlClocksEventReasonSyncBoost

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. Sync boost activated.

nvmlClocksThrottleReasonAll

Deprecated: All clock throttling reasons.

Deprecated:

Use *nvmlClocksEventReasonAll* instead

nvmlClocksThrottleReasonApplicationsClocksSetting

Deprecated: Application clocks setting.

Deprecated:

nvmlClocksThrottleReasonDisplayClockSetting

Deprecated: Display clock setting.

Deprecated:

Use *nvmlClocksEventReasonDisplayClockSetting* instead

nvmlClocksThrottleReasonGpuIdle

Deprecated: GPU idle.

Deprecated:

Use *nvmlClocksEventReasonGpuIdle* instead

nvmlClocksThrottleReasonHwPowerBrakeSlowdown

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

This is an indicator of:

- External Power Brake Assertion being triggered (e.g. by the system power supply)

See also

nvmlDeviceGetTemperature

See also

nvmlDeviceGetTemperatureThreshold

See also

nvmlDeviceGetPowerUsage Hardware power brake slowdown activated.

nvmlClocksThrottleReasonHwSlowdown

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

See also

nvmlDeviceGetTemperatureThreshold

See also

nvmlDeviceGetPowerUsage Hardware slowdown activated.

`nvmlClocksThrottleReasonHwThermalSlowdown`

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

This is an indicator of:

- temperature being too high

See also

nvmlDeviceGetTemperature

See also

nvmlDeviceGetTemperatureThreshold

See also

nvmlDeviceGetPowerUsage Hardware thermal slowdown activated.

`nvmlClocksThrottleReasonNone`

Deprecated: No clock throttling.

Deprecated:

Use *nvmlClocksEventReasonNone* instead

nvmlClocksThrottleReasonSwPowerCap

Deprecated: Software power cap.

Deprecated:

Use *nvmlClocksEventReasonSwPowerCap* instead

nvmlClocksThrottleReasonSwThermalSlowdown

Deprecated: Software thermal slowdown.

Deprecated:

Use *nvmlClocksEventReasonSwThermalSlowdown* instead

nvmlClocksThrottleReasonSyncBoost

Deprecated: Sync boost.

Deprecated:

Use *nvmlClocksEventReasonSyncBoost* instead

nvmlClocksThrottleReasonUserDefinedClocks

Deprecated: User-defined clocks.

Deprecated:

Renamed to *nvmlClocksThrottleReasonApplicationsClocksSetting* as the name describes the situation more accurately.

5.2.1.25 PRM Access

This chapter describes NVML operations that are associated with PRM register reads.

5.2.1.25.1 Macros

NVML_PRM_DATA_MAX_SIZE

Maximum size of the PRM data.

5.2.1.25.2 Enumerations

nvmlPRMCounterId_t

PRM Counter IDs.

5.2.1.25.3 Functions

nvmlReturn_t *nvmlDeviceReadPRMCounters_v1*(nvmlDevice_t device, nvmlPRMCounterList_v1_t *counterList)
Read a list of GPU PRM Counters.

nvmlReturn_t *nvmlDeviceReadWritePRM_v1*(nvmlDevice_t device, nvmlPRMTLV_v1_t *buffer)
Read or write a GPU PRM register.

5.2.1.25.4 Structs

nvmlPRMCounterInput_v1_t
PRM counter input values.

nvmlPRMCounterList_v1_t
PRM Counter List Structure v1.

nvmlPRMCounterValue_v1_t
PRM Counter Value Structure.

nvmlPRMCounter_v1_t
PRM Counter Structure v1.

nvmlPRMTLV_v1_t
Main PRM input structure.

5.2.1.25.5 Macros

NVML_PRM_DATA_MAX_SIZE
Maximum size of the PRM data.

5.2.1.25.6 Enumerations

enum **nvmlPRMCounterId_t**
PRM Counter IDs.

Values:

enumerator **NVML_PRM_COUNTER_ID_NONE**
Sentinel.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PHYSICAL_LAYER_CTRS_LINK_DOWN_EVENTS**
PPCNT group 0x12, link_down_events.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PHYSICAL_LAYER_CTRS_SUCCESSFUL_RECOVERY_EVENTS**
PPCNT group 0x12, successful_recovery_events.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_SUCCESSFUL_RECOVERY_EVENTS**
PPCNT group 0x1A, total_successful_recovery_events.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TIME_SINCE_LAST_RECOVERY**

PPCNT group 0x1A, time_since_last_recovery.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TIME_BETWEEN_LAST_TWO_RECOVERIES**

PPCNT group 0x1A, time_between_last_two_recoveries.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TIME_IN_LAST_HOST_SERDES_FEQ_RECOVERY**

PPCNT group 0x1A, time_in_last_host_serdes_feq_recovery.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_TIME_IN_HOST_SERDES_FEQ_RECOVERY**

PPCNT group 0x1A, total_time_in_host_serdes_feq_recovery.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_HOST_SERDES_FEQ_RECOVERY_COUNT**

PPCNT group 0x1A, total_host_serdes_feq_recovery_count.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_HOST_SERDES_FEQ_SUCCESSFUL_RECOVERY_COUN**

PPCNT group 0x1A, total_host_serdes_feq_successful_recovery_count.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_LAST_HOST_SERDES_FEQ_ATTEMPTS_COUNT**

PPCNT group 0x1A, last_host_serdes_feq_attempts_count.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_LAST_SUCCESSFUL_RECOVERY_STEP_ATTEMPTS**

PPCNT group 0x1A, last_successful_recovery_step_attempts.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_LAST_SUCCESSFUL_RECOVERY_TIME**

PPCNT group 0x1A, last_successful_recovery_time.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_SUCCESSFUL_RECOVERY_TIME**

PPCNT group 0x1A, total_successful_recovery_time.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PORTCOUNTERS_PORT_XMIT_WAIT**

PPCNT group 0x20, port_xmit_wait.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PLR_RCV_CODES**

PPCNT group 0x22, plr_rcv_codes.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PLR_RCV_CODE_ERR**

PPCNT group 0x22, plr_rcv_code_err.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PLR_RCV_UNCORRECTABLE_CODE**
PPCNT group 0x22, plr_rcv_uncorrectable_code.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PLR_XMIT_CODES**
PPCNT group 0x22, plr_xmit_codes.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PLR_XMIT_RETRY_CODES**
PPCNT group 0x22, plr_xmit_retry_codes.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PLR_XMIT_RETRY_EVENTS**
PPCNT group 0x22, plr_xmit_retry_events.

enumerator **NVML_PRM_COUNTER_ID_PPCNT_PLR_SYNC_EVENTS**
PPCNT group 0x22, plr_sync_events.

enumerator **NVML_PRM_COUNTER_ID_PPRM_OPER_RECOVERY**
PPRM, oper_recovery.

5.2.1.25.7 Functions

nvmlReturn_t **nvmlDeviceReadPRMCounters_v1**(
nvmlDevice_t device, *nvmlPRMCounterList_v1_t* *counterList,
)

Read a list of GPU PRM Counters.

For Blackwell or newer fully supported devices.

Supported on Linux only.

Parameters

- **device** – Identifier of target GPU device
- **counterList** – Structure holding the input parameters as well as the retrieved counter values

Returns

- *NVML_SUCCESS* on success
- *NVML_ERROR_INVALID_ARGUMENT* if `device` is invalid or `counterList` is NULL
- *NVML_ERROR_NO_PERMISSION* if user does not have permission to perform this operation
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by the device
- *NVML_ERROR_UNKNOWN* on any other error

nvmlReturn_t **nvmlDeviceReadWritePRM_v1**(
nvmlDevice_t device, *nvmlPRMTLV_v1_t* *buffer,
)

Read or write a GPU PRM register.

The input is assumed to be in TLV format in network byte order.

For Blackwell or newer fully supported devices.

Supported on Linux only.

Parameters

- **`device`** – Identifier of target GPU device
- **`buffer`** – Structure holding the input data in TLV format as well as the PRM register contents in TLV format (in the case of a successful read operation). Note: the input data and any returned data shall be in network byte order.

Returns

- *NVML_SUCCESS* on success
- *NVML_ERROR_INVALID_ARGUMENT* if `device` or `buffer` are invalid
- *NVML_ERROR_NO_PERMISSION* if user does not have permission to perform this operation
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by the device

5.2.1.26 Power Profile Information

5.2.1.26.1 Macros

NVML_WORKLOAD_POWER_MAX_PROFILES

nvmlWorkloadPowerProfileCurrentProfiles_v1

nvmlWorkloadPowerProfileInfo_v1
Version macro for *nvmlWorkloadPowerProfileInfo_v1_t* .

nvmlWorkloadPowerProfileProfilesInfo_v1
Version macro for *nvmlWorkloadPowerProfileProfilesInfo_v1_t* .

nvmlWorkloadPowerProfileRequestedProfiles_v1
Version macro for *nvmlWorkloadPowerProfileRequestedProfiles_v1_t* .

nvmlWorkloadPowerProfileUpdateProfiles_v1

5.2.1.26.2 Enumerations

nvmlPowerProfileOperation_t
Enum for operation to perform on the requested profiles.

nvmlPowerProfileType_t
Workload power profile identifiers.

5.2.1.26.3 Functions

nvmlReturn_t *nvmlDeviceWorkloadPowerProfileClearRequestedProfiles*(nvmlDevice_t device, nvmlWorkloadPowerProfileRequestedProfiles_t *requestedProfiles)

nvmlReturn_t *nvmlDeviceWorkloadPowerProfileGetCurrentProfiles*(nvmlDevice_t device, nvmlWorkloadPowerProfileCurrentProfiles_t *currentProfiles)
Get Current Performance Profiles.

nvmlReturn_t *nvmlDeviceWorkloadPowerProfileGetProfilesInfo*(nvmlDevice_t device, nvmlWorkloadPowerProfileProfilesInfo_t *profilesInfo)
Get Performance Profiles Information.

nvmlReturn_t *nvmlDeviceWorkloadPowerProfileSetRequestedProfiles*(nvmlDevice_t device, nvmlWorkloadPowerProfileRequestedProfiles_t *requestedProfiles)

nvmlReturn_t *nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1*(nvmlDevice_t device, nvmlWorkloadPowerProfileUpdateProfiles_v1_t *updateProfiles)
Update Requested Performance Profiles.

5.2.1.26.4 Structs

nvmlWorkloadPowerProfileCurrentProfiles_v1_t
Current Profiles.

nvmlWorkloadPowerProfileInfo_v1_t
Profile Metadata.

nvmlWorkloadPowerProfileProfilesInfo_v1_t
Profiles Info.

nvmlWorkloadPowerProfileRequestedProfiles_v1_t
Requested Profiles.

nvmlWorkloadPowerProfileUpdateProfiles_v1_t
Update Profiles.

5.2.1.26.5 Typedefs

nvmlWorkloadPowerProfileCurrentProfiles_t

nvmlWorkloadPowerProfileInfo_t

nvmlWorkloadPowerProfileProfilesInfo_t

nvmlWorkloadPowerProfileRequestedProfiles_t

5.2.1.26.6 Macros

NVML_WORKLOAD_POWER_MAX_PROFILES

nvmlWorkloadPowerProfileCurrentProfiles_v1

nvmlWorkloadPowerProfileInfo_v1

Version macro for nvmlWorkloadPowerProfileInfo_v1_t.

nvmlWorkloadPowerProfileProfilesInfo_v1

Version macro for nvmlWorkloadPowerProfileProfilesInfo_v1_t.

nvmlWorkloadPowerProfileRequestedProfiles_v1

Version macro for nvmlWorkloadPowerProfileRequestedProfiles_v1_t.

nvmlWorkloadPowerProfileUpdateProfiles_v1

5.2.1.26.7 Enumerations

enum **nvmlPowerProfileOperation_t**

Enum for operation to perform on the requested profiles.

Values:

enumerator **NVML_POWER_PROFILE_OPERATION_CLEAR**

Remove the requested profiles from the existing list of requested profiles.

enumerator **NVML_POWER_PROFILE_OPERATION_SET**

Add the requested profiles to the existing list of requested profiles.

enumerator **NVML_POWER_PROFILE_OPERATION_SET_AND_OVERWRITE**

Overwrite the existing list of requested profiles with just the requested profiles.

enumerator **NVML_POWER_PROFILE_OPERATION_MAX**

Max value above +1.

enum **nvmlPowerProfileType_t**

Workload power profile identifiers.

Values:

enumerator **NVML_POWER_PROFILE_MAX_P**

enumerator **NVML_POWER_PROFILE_MAX_Q**

enumerator **NVML_POWER_PROFILE_COMPUTE**

enumerator **NVML_POWER_PROFILE_MEMORY_BOUND**

enumerator **NVML_POWER_PROFILE_NETWORK**

enumerator **NVML_POWER_PROFILE_BALANCED**

enumerator **NVML_POWER_PROFILE_LLM_INFERENCE**

enumerator **NVML_POWER_PROFILE_LLM_TRAINING**

enumerator **NVML_POWER_PROFILE_RBM**

enumerator **NVML_POWER_PROFILE_DCPCIE**

enumerator **NVML_POWER_PROFILE_HMMA_SPARSE**

enumerator **NVML_POWER_PROFILE_HMMA_DENSE**

enumerator **NVML_POWER_PROFILE_SYNC_BALANCED**

enumerator **NVML_POWER_PROFILE_HPC**

enumerator **NVML_POWER_PROFILE_MIG**

enumerator **NVML_POWER_PROFILE_MAX_Q_1**

enumerator **NVML_POWER_PROFILE_NETWORK_BOUND**

enumerator **NVML_POWER_PROFILE_HIGH_THROUGHPUT_INFERENCE**

enumerator **NVML_POWER_PROFILE_MEDIUM_THROUGHPUT_INFERENCE**

enumerator **NVML_POWER_PROFILE_LOW_LATENCY_INFERENCE**

enumerator **NVML_POWER_PROFILE_TRAINING**

enumerator **NVML_POWER_PROFILE_INFERENCE**

enumerator **NVML_POWER_PROFILE_MAX_Q_2**

enumerator **NVML_POWER_PROFILE_MAX_Q_3**

enumerator **NVML_POWER_PROFILE_LOW_PRIORITY_BACKGROUND**

enumerator **NVML_POWER_PROFILE_MAX**

5.2.1.26.8 Functions

nvmlReturn_t **nvmlDeviceWorkloadPowerProfileClearRequestedProfiles**(
nvmlDevice_t device, *nvmlWorkloadPowerProfileRequestedProfiles_t* *requestedProfiles,
)

Deprecated:

Use *nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1* instead Clear Requested Performance Profiles

For Blackwell or newer fully supported devices. See *nvmlWorkloadPowerProfileRequestedProfiles_v1_t* for more information on the struct. Clear one or more performance profiles be using the input bitmask *requestedProfilesMask*, where each bit set corresponds to a supported bit from the *perfProfilesMask*. These profiles will be removed from the existing list of currently requested profiles. Requires root/admin permissions.

Parameters

- **device** – The identifier of the target device
- **requestedProfiles** – Reference to struct nvmlWorkloadPowerProfileRequestedProfiles_v1_t

Returns

- *NVML_SUCCESS* If the query is successful
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid or *pointer* to struct is NULL
- *NVML_ERROR_NOT_SUPPORTED* If the device does not support this feature
- *NVML_ERROR_GPU_IS_LOST* If the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the provided version is invalid/unsupported
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceWorkloadPowerProfileGetCurrentProfiles**(
nvmlDevice_t device, *nvmlWorkloadPowerProfileCurrentProfiles_t* *currentProfiles,
)

Get Current Performance Profiles.

For Blackwell or newer fully supported devices. See *nvmlWorkloadPowerProfileCurrentProfiles_v1_t* for more information on the struct. This API returns a stuct which contains the current *perfProfilesMask*, *requestedProfilesMask* and *enforcedProfilesMask*. Each bit set in each bitmasks indicates the profile is supported, currently requested or currently engaged, respectively.

Parameters

- **device** – The identifier of the target device
- **currentProfiles** – Reference to struct nvmlWorkloadPowerProfileCurrentProfiles_v1_t

Returns

- *NVML_SUCCESS* If the query is successful
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid or the pointer to struct is NULL
- *NVML_ERROR_NOT_SUPPORTED* If the device does not support this feature
- *NVML_ERROR_GPU_IS_LOST* If the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the provided version is invalid/unsupported
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceWorkloadPowerProfileGetProfilesInfo**(
nvmlDevice_t device, *nvmlWorkloadPowerProfileProfilesInfo_t* *profilesInfo,
)

Get Performance Profiles Information.

For Blackwell or newer fully supported devices. See *nvmlWorkloadPowerProfileProfilesInfo_v1_t* for more information on the struct. The mask *perfProfilesMask* is bitmask of all supported mode indices where the mode is supported if the index is 1. Each supported mode will have a corresponding entry in the *perfProfile* array which will contain the *profileId*, the *priority* of this mode, where the lower the value, the higher the priority, and a *conflictingMask*, where each bit set in the mask corresponds to a different profile which cannot be used in conjunction with the given profile.

Parameters

- **device** – The identifier of the target device
- **profilesInfo** – Reference to struct *nvmlWorkloadPowerProfileProfilesInfo_t*

Returns

- *NVML_SUCCESS* If the query is successful
- *NVML_ERROR_INSUFFICIENT_SIZE* If struct is fully allocated
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid or *pointer* to struct is NULL
- *NVML_ERROR_NOT_SUPPORTED* If the device does not support this feature
- *NVML_ERROR_GPU_IS_LOST* If the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the provided version is invalid/unsupported
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceWorkloadPowerProfileSetRequestedProfiles**(
nvmlDevice_t device, *nvmlWorkloadPowerProfileRequestedProfiles_t* *requestedProfiles,
)

Deprecated:

Use *nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1* instead Set Requested Performance Profiles

For Blackwell or newer fully supported devices. See *nvmlWorkloadPowerProfileRequestedProfiles_v1_t* for more information on the struct. Request one or more performance profiles be activated using the input bitmask *requestedProfilesMask*, where each bit set corresponds to a supported bit from the *perfProfilesMask*. These profiles will be added to existing list of currently requested profiles. Requires root/admin permissions.

Parameters

- **device** – The identifier of the target device
- **requestedProfiles** – Reference to struct nvmlWorkloadPowerProfileRequestedProfiles_v1_t

Returns

- *NVML_SUCCESS* If the query is successful
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid or *pointer* to struct is NULL
- *NVML_ERROR_NOT_SUPPORTED* If the device does not support this feature
- *NVML_ERROR_GPU_IS_LOST* If the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the provided version is invalid/unsupported
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1**(
nvmlDevice_t device, *nvmlWorkloadPowerProfileUpdateProfiles_v1_t* *updateProfiles,
)

Update Requested Performance Profiles.

For Blackwell or newer fully supported devices. See *nvmlWorkloadPowerProfileUpdateProfiles_v1_t* for more information on the struct. Update the requested performance profiles using the input bitmask *updateProfilesMask*, where each bit set corresponds to a supported bit from the *perfProfilesMask*. The *operation* parameter specifies the operation to perform, see *nvmlPowerProfileOperation_t* for more information. Requires root/admin permissions or access to the NVIDIA WPPS capability.

Parameters

- **device** – The identifier of the target device
- **updateProfiles** – Reference to struct nvmlWorkloadPowerProfileUpdateProfiles_v1_t

Returns

- *NVML_SUCCESS* If the query is successful
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid or *pointer* to struct 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

5.2.1.26.9 Typedefs

typedef *nvmlWorkloadPowerProfileCurrentProfiles_v1_t*
nvmlWorkloadPowerProfileCurrentProfiles_t

typedef *nvmlWorkloadPowerProfileInfo_v1_t* **nvmlWorkloadPowerProfileInfo_t**

typedef *nvmlWorkloadPowerProfileProfilesInfo_v1_t* **nvmlWorkloadPowerProfileProfilesInfo_t**

typedef *nvmlWorkloadPowerProfileRequestedProfiles_v1_t*
nvmlWorkloadPowerProfileRequestedProfiles_t

5.2.1.27 Power Smoothing Information

5.2.1.27.1 Macros

NVML_POWER_SMOOTHING_ADMIN_OVERRIDE_NOT_SET
Admin override not set.

NVML_POWER_SMOOTHING_IDX_FROM_FIELD_VAL
Deprecated, do not use.

NVML_POWER_SMOOTHING_IDX_FROM_FIELD_VAL_DEPRECATED
Index from field value.

NVML_POWER_SMOOTHING_MAX_NUM_PROFILES
Maximum number of profiles.

NVML_POWER_SMOOTHING_NUM_PROFILE_PARAMS
Number of profile parameters.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PERCENT_TMP_FLOOR
Percent temperature floor.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_ACT_OFFSET
Primary floor activation offset value for a given profile.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_ACT_WIN_MULT
Primary floor activation window multiplier value for a given profile.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_TAR_WIN_MULT
Primary floor target window multiplier value for a given profile.

NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_DOWN_HYSTERESIS
Ramp down hysteresis.

NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_DOWN_RATE
Ramp down rate.

NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_UP_RATE
Ramp up rate.

NVML_POWER_SMOOTHING_PROFILE_PARAM_SECONDARY_POWER_FLOOR
Secondary power floor value for a given profile.

nvmlPowerSmoothingProfile_v1
Version macro for *nvmlPowerSmoothingProfile_v1_t* .

nvmlPowerSmoothingState_v1
Version macro for *nvmlPowerSmoothingState_v1_t* .

5.2.1.27.2 Functions

nvmlReturn_t *nvmlDevicePowerSmoothingActivatePresetProfile***(nvmlDevice_t device, nvmlPowerSmoothingProfile_t *profile)**
Activate a specific preset profile for datacenter power smoothing.

nvmlReturn_t *nvmlDevicePowerSmoothingSetState***(nvmlDevice_t device, nvmlPowerSmoothingState_t *state)**
Enable or disable the Power Smoothing Feature.

nvmlReturn_t *nvmlDevicePowerSmoothingUpdatePresetProfileParam***(nvmlDevice_t device, nvmlPowerSmoothingProfile_t *profile)**
Update the value of a specific profile parameter contained within *nvmlPowerSmoothingProfile_v1_t* .

5.2.1.27.3 Structs

nvmlPowerSmoothingProfile_v1_t
Power Smoothing Structure for Profile information.

nvmlPowerSmoothingState_v1_t
Power Smoothing Structure for Feature Enablement.

5.2.1.27.4 Typedefs

nvmlPowerSmoothingProfile_t

nvmlPowerSmoothingState_t

5.2.1.27.5 Macros

NVML_POWER_SMOOTHING_ADMIN_OVERRIDE_NOT_SET
Admin override not set.

NVML_POWER_SMOOTHING_IDX_FROM_FIELD_VAL(field_val)
Deprecated, do not use.

NVML_POWER_SMOOTHING_IDX_FROM_FIELD_VAL_DEPRECATED(field_val)
Index from field value.

NVML_POWER_SMOOTHING_MAX_NUM_PROFILES
Maximum number of profiles.

NVML_POWER_SMOOTHING_NUM_PROFILE_PARAMS
Number of profile parameters.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PERCENT_TMP_FLOOR
Percent temperature floor.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_ACT_OFFSET
Primary floor activation offset value for a given profile.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_ACT_WIN_MULT
Primary floor activation window multiplier value for a given profile.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_TAR_WIN_MULT
Primary floor target window multiplier value for a given profile.

NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_DOWN_HYSTERESIS
Ramp down hysteresis.

NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_DOWN_RATE
Ramp down rate.

NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_UP_RATE
Ramp up rate.

NVML_POWER_SMOOTHING_PROFILE_PARAM_SECONDARY_POWER_FLOOR
Secondary power floor value for a given profile.

nvmlPowerSmoothingProfile_v1
Version macro for nvmlPowerSmoothingProfile_v1_t.

nvmlPowerSmoothingState_v1
Version macro for nvmlPowerSmoothingState_v1_t.

5.2.1.27.6 Functions

nvmlReturn_t **nvmlDevicePowerSmoothingActivatePresetProfile**(
nvmlDevice_t device, *nvmlPowerSmoothingProfile_t* *profile,
)

Activate a specific preset profile for datacenter power smoothing.

The API only sets the active preset profile based on the input profileId, and ignores the other parameters of the structure. Requires root/admin permissions.

For Blackwell or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **profile** – Reference to *nvmlPowerSmoothingProfile_v1_t*. Note that only *profile->profileId* is used and the rest of the structure is ignored.

Returns

- *NVML_SUCCESS* if the Desired Profile was successfully set
- *NVML_ERROR_INVALID_ARGUMENT* if device is invalid or structure was NULL
- *NVML_ERROR_NO_PERMISSION* if user does not have permission to change the profile number
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by the device

nvmlReturn_t **nvmlDevicePowerSmoothingSetState**(
nvmlDevice_t device, *nvmlPowerSmoothingState_t* *state,
)

Enable or disable the Power Smoothing Feature.

Requires root/admin permissions.

For Blackwell or newer fully supported devices.

See *nvmlEnableState_t* for details on allowed states

Parameters

- **device** – The identifier of the target device
- **state** – Reference to *nvmlPowerSmoothingState_v1_t*

Returns

- *NVML_SUCCESS* if the feature state was successfully set
- *NVML_ERROR_INVALID_ARGUMENT* if device is invalid or state is NULL
- *NVML_ERROR_NO_PERMISSION* if user does not have permission to change feature state
- *NVML_ERROR_NOT_SUPPORTED* if this feature is not supported by the device

nvmlReturn_t **nvmlDevicePowerSmoothingUpdatePresetProfileParam**(
nvmlDevice_t device, *nvmlPowerSmoothingProfile_t* *profile,
)

Update the value of a specific profile parameter contained within *nvmlPowerSmoothingProfile_v1_t*.

Requires root/admin permissions.

For Blackwell or newer fully supported devices.

NVML_POWER_SMOOTHING_PROFILE_PARAM_PERCENT_TMP_FLOOR expects a value as a percentage from 00.00-100.00% NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_UP_RATE expects a value in mW/s NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_DOWN_RATE expects a value in mW/s NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_DOWN_HYSTERESIS expects a value in ms NVML_POWER_SMOOTHING_PROFILE_PARAM_SECONDARY_POWER_FLOOR

expects a value in mW NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_ACT_WIN_MULT expects a value from 0-255 NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_TAR_WIN_MULT expects a value from 0-255 NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_ACT_OFFSET expects a value in mW

Parameters

- **device** – The identifier of the target device
- **profile** – Reference to *nvmlPowerSmoothingProfile_v1_t* struct

Returns

- *NVML_SUCCESS* if the Active Profile was successfully set
- *NVML_ERROR_INVALID_ARGUMENT* if device is invalid or profile parameter/value was invalid
- *NVML_ERROR_NO_PERMISSION* if user does not have permission to change any profile parameters
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* if the structure version is not supported

5.2.1.27.7 Typedefs

typedef *nvmlPowerSmoothingProfile_v1_t* **nvmlPowerSmoothingProfile_t**

typedef *nvmlPowerSmoothingState_v1_t* **nvmlPowerSmoothingState_t**

5.2.1.28 System Queries

This chapter describes the queries that NVML can perform against the local system.

These queries are not device-specific.

5.2.1.28.1 Macros

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

NVML_CUDA_DRIVER_VERSION_MINOR
Macro to extract the minor version number from the CUDA driver version.

nvmlSystemDriverBranchInfo_v1
Version macro for *nvmlSystemDriverBranchInfo_v1_t* .

5.2.1.28.2 Functions

nvmlReturn_t *nvmlSystemGetCPER_v1*(nvmlGetCPER_v1_t *cper)
Retrieves Common Platform Error Record (CPER) data.

nvmlReturn_t *nvmlSystemGetCudaDriverVersion*(int *cudaDriverVersion)
Retrieves the version of the CUDA driver.

nvmlReturn_t *nvmlSystemGetCudaDriverVersion_v2*(int *cudaDriverVersion)
Retrieves the version of the CUDA driver from the shared library.

nvmlReturn_t *nvmlSystemGetDriverBranch*(nvmlSystemDriverBranchInfo_t *branchInfo, unsigned int length)
Retrieves the driver branch of the NVIDIA driver installed on the system.

nvmlReturn_t *nvmlSystemGetDriverVersion*(char *version, unsigned int length)
Retrieves the version of the system's graphics driver.

nvmlReturn_t *nvmlSystemGetHicVersion*(unsigned int *hwbcCount, nvmlHwbcEntry_t *hwbcEntries)
Retrieves the IDs and firmware versions for any Host Interface Cards (HICs) in the system.

nvmlReturn_t *nvmlSystemGetNVMLVersion*(char *version, unsigned int length)
Retrieves the version of the NVML library.

nvmlReturn_t *nvmlSystemGetProcessName*(unsigned int pid, char *name, unsigned int length)
Gets name of the process with provided process id.

nvmlReturn_t *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.

5.2.1.28.3 Structs

nvmlSystemDriverBranchInfo_v1_t
Structure to store Driver branch information.

5.2.1.28.4 Typedefs

nvmlSystemDriverBranchInfo_t

5.2.1.28.5 Macros

NVML_CUDA_DRIVER_VERSION_MAJOR(v)
Macros for converting the CUDA driver version number to Major and Minor version numbers.

Macro to extract the major version number from the CUDA driver version.

NVML_CUDA_DRIVER_VERSION_MINOR(v)
Macro to extract the minor version number from the CUDA driver version.

nvmlSystemDriverBranchInfo_v1
Version macro for nvmlSystemDriverBranchInfo_v1_t.

5.2.1.28.6 Functions

nvmlReturn_t **nvmlSystemGetCPER_v1**(*nvmlGetCPER_v1_t* *cper)
Retrieves Common Platform Error Record (CPER) data.

Records are returned in a caller-supplied buffer. Iteration is driven by the *cursor* (*nvmlCPERCursor_v1_t*) struct: pass the same *cursor* on every call in a sequence; the implementation updates *cursor.handle*. Do not modify *cursor* between calls. To change *cursor.cperTypeMask* or *cursor.uuid*, set *cursor.handle* to *NVML_CPER_CURSOR_HANDLE_INIT* and call again (new iteration).

For a size query, call with *buffer* NULL and *bufferSize* 0; the function returns *NVML_ERROR_INSUFFICIENT_SIZE* and sets *bufferSize* when records exist, or *NVML_SUCCESS* with *bufferSize* set to 0 when there are no CPER records. Use *bufferSize* == 0 on return as the indicator for "no records" or "no more records".

This API requires root privileges. Records are available from initialization.

Parameters

cper – Pointer to an *nvmlGetCPER_v1_t*. On entry set *cursor.cperTypeMask*, *cursor.uuid* (empty string for all), *cursor.handle* (to *NVML_CPER_CURSOR_HANDLE_INIT* for first call), *buffer* (or NULL), *bufferSize*. On return *cursor.handle* and *bufferSize* are updated.

Returns

- *NVML_SUCCESS* Buffer has been populated with CPER data, or *bufferSize* is 0 (no/more records).
- *NVML_ERROR_UNINITIALIZED* The library has not been successfully initialized.
- *NVML_ERROR_INVALID_ARGUMENT* *cper* is NULL, or *buffer* is NULL while *bufferSize* is non-zero.
- *NVML_ERROR_NOT_SUPPORTED* The feature is not supported on this system.
- *NVML_ERROR_INSUFFICIENT_SIZE* Buffer too small; *bufferSize* set to required size.
- *NVML_ERROR_NO_PERMISSION* Insufficient privileges.
- *NVML_ERROR_UNKNOWN* An unexpected error occurred.

nvmlReturn_t **nvmlSystemGetCudaDriverVersion**(int *cudaDriverVersion)

Retrieves the version of the CUDA driver.

For all products.

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

Parameters

cudaDriverVersion – Reference in which to return the version identifier

Returns

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

nvmlReturn_t **nvmlSystemGetCudaDriverVersion_v2**(
int *cudaDriverVersion,
)

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

For all products.

The returned CUDA driver version by calling cuDriverGetVersion()

Parameters

cudaDriverVersion – Reference in which to return the version identifier

Returns

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

nvmlReturn_t **nvmlSystemGetDriverBranch**(
nvmlSystemDriverBranchInfo_t *branchInfo, unsigned int length,
)

Retrieves the driver branch of the NVIDIA driver installed on the system.

For all products.

The branch 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

- **branchInfo** – Pointer to the driver branch information structure *nvmlSystemDriverBranchInfo_t*
- **length** – The maximum allowed length of the driver branch string

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *branchInfo* is NULL
- *NVML_ERROR_INSUFFICIENT_SIZE* if *length* is too small
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **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

nvmlReturn_t **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

nvmlReturn_t **nvmlSystemGetNVMLVersion**(
char *version, unsigned int length,
)

Retrieves the version of the NVML library.

For all products.

Note

This query is available in the MPS locality-domain view described in *MPS Locality Domain Device Enumeration*. 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

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

nvmlReturn_t **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

5.2.1.28.7 Typedefs

typedef *nvmlSystemDriverBranchInfo_v1_t* **nvmlSystemDriverBranchInfo_t**

5.2.1.29 Unit Commands

This chapter describes NVML operations that change the state of the unit.

For S-class products. Each of these requires root/admin access. Non-admin users will see an NVML_ERROR_NO_PERMISSION error code when invoking any of these methods.

5.2.1.29.1 Functions

nvmlReturn_t *nvmlUnitSetLedState*(nvmlUnit_t unit, nvmlLedColor_t color)
Set the LED state for the unit.

5.2.1.29.2 Functions

nvmlReturn_t **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.

See also

nvmlUnitGetLedState()

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

5.2.1.30 Unit Queries

This chapter describes that queries that NVML can perform against each unit.

For S-class systems only. In each case the device is identified with an nvmlUnit_t handle. This handle is obtained by calling *nvmlUnitGetHandleByIndex()*.

5.2.1.30.1 Functions

nvmlReturn_t *nvmlUnitGetCount*(unsigned int *unitCount)
Retrieves the number of units in the system.

nvmlReturn_t *nvmlUnitGetDevices*(nvmlUnit_t unit, unsigned int *deviceCount, nvmlDevice_t *devices)
Retrieves the set of GPU devices that are attached to the specified unit.

nvmlReturn_t *nvmlUnitGetFanSpeedInfo*(nvmlUnit_t unit, nvmlUnitFanSpeeds_t *fanSpeeds)
Retrieves the fan speed readings for the unit.

nvmlReturn_t *nvmlUnitGetHandleByIndex*(unsigned int index, nvmlUnit_t *unit)
Acquire the handle for a particular unit, based on its index.

nvmlReturn_t *nvmlUnitGetLedState*(nvmlUnit_t unit, nvmlLedState_t *state)
Retrieves the LED state associated with this unit.

nvmlReturn_t *nvmlUnitGetPsuInfo*(nvmlUnit_t unit, nvmlPSUInfo_t *psu)
Retrieves the PSU stats for the unit.

nvmlReturn_t *nvmlUnitGetTemperature*(nvmlUnit_t unit, unsigned int type, unsigned int *temp)
Retrieves the temperature readings for the unit, in degrees C.

nvmlReturn_t *nvmlUnitGetUnitInfo*(nvmlUnit_t unit, nvmlUnitInfo_t *info)
Retrieves the static information associated with a unit.

5.2.1.30.2 Functions

nvmlReturn_t **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

nvmlReturn_t **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

nvmlReturn_t **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

nvmlReturn_t **nvmlUnitGetHandleByIndex**(
    unsigned int index, *nvmlUnit_t* *unit,
)

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

For S-class products.

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

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

Parameters

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

Returns

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

nvmlReturn_t **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.

See also

nvmlUnitSetLedState()

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

nvmlReturn_t **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

nvmlReturn_t **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

nvmlReturn_t **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

5.2.1.31 Unit Structs

5.2.1.31.1 Enumerations

nvmlFanState_t
Fan state enum.

nvmlLedColor_t
Led color enum.

5.2.1.31.2 Structs

nvmlHwbcEntry_t
Description of HWBC entry.

nvmlLedState_t
LED states for an S-class unit.

nvmlPSUInfo_t
Power usage information for an S-class unit.

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

nvmlUnitFanSpeeds_t
Fan speed readings for an entire S-class unit.

nvmlUnitInfo_t
Static S-class unit info.

5.2.1.31.3 Typedefs

nvmlUnit_t

5.2.1.31.4 Enumerations

enum **nvmlFanState_t**
Fan state enum.

Values:

enumerator **NVML_FAN_NORMAL**
Fan is working properly.

enumerator **NVML_FAN_FAILED**
Fan has failed.

enum **nvmlLedColor_t**
Led color enum.
Values:

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

enumerator **NVML_LED_COLOR_AMBER**
AMBER, indicates problem.

5.2.1.31.5 Typedefs

typedef struct nvmlUnit_st ***nvmlUnit_t**

5.2.1.32 vGPU APIs

This chapter describes operations that are associated with NVIDIA vGPU Software products.

5.2.1.32.1 Functions

nvmlReturn_t *nvmlDeviceGetGridLicensableFeatures_v4*(nvmlDevice_t device, nvmlGridLicensableFeatures_t *pGridLicensableFeatures)
Retrieve the vGPU Software licensable features.

nvmlReturn_t *nvmlDeviceGetHostVgpuMode*(nvmlDevice_t device, nvmlHostVgpuMode_t *pHostVgpuMode)
Queries if SR-IOV host operation is supported on a vGPU supported device.

nvmlReturn_t *nvmlDeviceGetVgpuHeterogeneousMode*(nvmlDevice_t device, nvmlVgpuHeterogeneousMode_t *pHeterogeneousMode)
Get the vGPU heterogeneous mode for the device.

nvmlReturn_t *nvmlDeviceGetVgpuTypeCreatablePlacements*(nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, nvmlVgpuPlacementList_t *pPlacementList)
Query the creatable vGPU placement ID of the vGPU type.

nvmlReturn_t *nvmlDeviceGetVgpuTypeSupportedPlacements*(nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, nvmlVgpuPlacementList_t *pPlacementList)
Query the supported vGPU placement ID of the vGPU type.

nvmlReturn_t *nvmlDeviceGetVirtualizationMode*(nvmlDevice_t device, nvmlGpuVirtualizationMode_t *pVirtualMode)
This method is used to get the virtualization mode corresponding to the GPU.

nvmlReturn_t *nvmlDeviceSetVgpuCapabilities*(nvmlDevice_t device, nvmlDeviceVgpuCapability_t capability, nvmlEnableState_t state)
Set the desirable vGPU capability of a device.

nvmlReturn_t *nvmlDeviceSetVgpuHeterogeneousMode*(nvmlDevice_t device, const nvmlVgpuHeterogeneousMode_t *pHeterogeneousMode)
Enable or disable vGPU heterogeneous mode for the device.

nvmlReturn_t *nvmlDeviceSetVirtualizationMode*(nvmlDevice_t device, nvmlGpuVirtualizationMode_t virtualMode)
This method is used to set the virtualization mode corresponding to the GPU.

nvmlReturn_t *nvmlDeviceVgpuForceGspUnload*(nvmlDevice_t device)
Executes a forced GSP unload operation on a device.

nvmlReturn_t *nvmlVgpuInstanceGetPlacementId*(nvmlVgpuInstance_t vgpuInstance, nvmlVgpuPlacementId_t *pPlacement)
Query the placement ID of active vGPU instance.

nvmlReturn_t *nvmlVgpuInstanceGetRuntimeStateSize*(nvmlVgpuInstance_t vgpuInstance, nvmlVgpuRuntimeState_t *pState)
Retrieve the currently used runtime state size of the vGPU instance.

nvmlReturn_t *nvmlVgpuTypeGetFbReservation*(nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *fbReservation)
Retrieve the static framebuffer reservation of the vGPU type in bytes.

nvmlReturn_t *nvmlVgpuTypeGetGspHeapSize*(nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *gspHeapSize)
Retrieve the static GSP heap size of the vGPU type in bytes.

5.2.1.32.2 Functions

nvmlReturn_t **nvmlDeviceGetGridLicensableFeatures_v4**(
nvmlDevice_t device, *nvmlGridLicensableFeatures_t* *pGridLicensableFeatures,
)

Retrieve the vGPU Software licensable features.

Identifies whether the system supports vGPU Software Licensing. If it does, return the list of licensable feature(s) and their current license status.

Parameters

- **device** – Identifier of the target device
- **pGridLicensableFeatures** – Pointer to structure in which vGPU software licensable features are returned

Returns

- *NVML_SUCCESS* if licensable features are successfully retrieved
- *NVML_ERROR_INVALID_ARGUMENT* if *pGridLicensableFeatures* is NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetHostVgpuMode**(
nvmlDevice_t device, *nvmlHostVgpuMode_t* *pHostVgpuMode,
)

Queries if SR-IOV host operation is supported on a vGPU supported device.

Checks whether SR-IOV host capability is supported by the device and the driver, and indicates device is in SR-IOV mode if both of these conditions are true.

Parameters

- **device** – The identifier of the target device
- **pHostVgpuMode** – Reference in which to return the current vGPU mode

Returns

- *NVML_SUCCESS* if device's vGPU mode has been successfully retrieved
- *NVML_ERROR_INVALID_ARGUMENT* if *device* handle is 0 or *pVgpuMode* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if *device* doesn't support this feature.
- *NVML_ERROR_UNKNOWN* if any unexpected error occurred

nvmlReturn_t **nvmlDeviceGetVgpuHeterogeneousMode**(
nvmlDevice_t device, *nvmlVgpuHeterogeneousMode_t* *pHeterogeneousMode,
)

Get the vGPU heterogeneous mode for the device.

When in heterogeneous mode, a vGPU can concurrently host timesliced vGPUs with differing framebuffer sizes.

On successful return, the function returns *pHeterogeneousMode->mode* with the current vGPU heterogeneous mode. *pHeterogeneousMode->version* is the version number of the structure nvmlVgpuHeterogeneousMode_t, the caller should set the correct version number to retrieve the vGPU heterogeneous mode. *pHeterogeneousMode->mode* can either be *NVML_FEATURE_ENABLED* or *NVML_FEATURE_DISABLED*.

Parameters

- **device** – The identifier of the target device
- **pHeterogeneousMode** – Pointer to the caller-provided structure of nvmlVgpuHeterogeneousMode_t

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid or *pHeterogeneousMode* is NULL
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* doesn't support this feature
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pHeterogeneousMode* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuTypeCreatablePlacements**(
nvmlDevice_t device, *nvmlVgpuTypeId_t* vgpuTypeId, *nvmlVgpuPlacementList_t* *pPlacementList,
)

Query the creatable vGPU placement ID of the vGPU type.

An array of creatable vGPU placement IDs for the vGPU type ID indicated by *vgpuTypeId* is returned in the caller-supplied buffer of *pPlacementList->placementIds*. Memory needed for the placementIds array should be allocated based on maximum instances of a vGPU type which can be queried via *nvmlVgpuTypeGetMaxInstances()*. If the provided count by the caller is insufficient, the function will return NVML_ERROR_INSUFFICIENT_SIZE along with the number of required entries in *pPlacementList->count*. The caller should then reallocate a buffer with the size of pPlacementList->count * sizeof(pPlacementList->placementIds) and invoke the function again.

The creatable vGPU placement IDs may differ over time, as there may be restrictions on what type of vGPU the vGPU instance is running.

Parameters

- **device** – The identifier of the target device
- **vgpuTypeId** – Handle to vGPU type. The vGPU type ID
- **pPlacementList** – Pointer to the list of vGPU placement structure *nvmlVgpuPlacementList_t*

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* or *vgpuTypeId* is invalid or *pPlacementList* is NULL
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* or *vgpuTypeId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pPlacementList* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuTypeSupportedPlacements**(
nvmlDevice_t device, *nvmlVgpuTypeId_t* vgpuTypeId, *nvmlVgpuPlacementList_t* *pPlacementList,
)

Query the supported vGPU placement ID of the vGPU type.

The function returns an array of supported vGPU placement IDs for the specified vGPU type ID in the buffer provided by the caller at *pPlacementList->placementIds*. The required memory for the placementIds array must be allocated based on the maximum number of vGPU type instances, which is retrievable through *nvmlVgpuTypeGetMaxInstances()*. If the provided count by the caller is insufficient, the function will return NVML_ERROR_INSUFFICIENT_SIZE along with the number of required entries in *pPlacementList->count*. The caller should then reallocate a buffer with the size of pPlacementList->count * sizeof(pPlacementList->placementIds) and invoke the function again.

To obtain a list of homogeneous placement IDs, the caller needs to set *pPlacementList->mode* to NVML_VGPU_PGPU_HOMOGENEOUS_MODE. For heterogeneous placement IDs, *pPlacementList->mode* should be set to NVML_VGPU_PGPU_HETEROGENEOUS_MODE. By default, a list of heterogeneous placement IDs is returned.

Parameters

- **device** – Identifier of the target device
- **vgpuTypeId** – Handle to vGPU type. The vGPU type ID
- **pPlacementList** – Pointer to the vGPU placement structure *nvmlVgpuPlacementList_t*

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* or *vgpuTypeId* is invalid or *pPlacementList* is NULL
- *NVML_ERROR_NOT_SUPPORTED* If *device* or *vgpuTypeId* isn't supported
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pPlacementList* is invalid
- *NVML_ERROR_INSUFFICIENT_SIZE* If the buffer is small, element count is returned in *pPlacementList->count*
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVirtualizationMode**(
    nvmlDevice_t device, *nvmlGpuVirtualizationMode_t* *pVirtualMode,
)

This method is used to get the virtualization mode corresponding to the GPU.

For Kepler or newer fully supported devices.

Parameters

- **device** – Identifier of the target device
- **pVirtualMode** – Reference to virtualization mode. One of NVML_GPU_VIRTUALIZATION_?

Returns

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

nvmlReturn_t **nvmlDeviceSetVgpuCapabilities**(
    nvmlDevice_t device, *nvmlDeviceVgpuCapability_t* capability, *nvmlEnableState_t* state,
)

Set the desirable vGPU capability of a device.

Refer to the *nvmlDeviceVgpuCapability_t* structure for the specific capabilities that can be set. See *nvmlEnableState_t* for available state.

Parameters

- **device** – The identifier of the target device
- **capability** – Specifies the *nvmlDeviceVgpuCapability_t* to be set
- **state** – The target capability mode

Returns

- *NVML_SUCCESS* Successful completion
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid, or *capability* is invalid, or *state* is invalid
- *NVML_ERROR_NOT_SUPPORTED* The API is not supported in current state, or *device* not in vGPU mode
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceSetVgpuHeterogeneousMode**(
nvmlDevice_t device, const *nvmlVgpuHeterogeneousMode_t* *pHeterogeneousMode,
)

Enable or disable vGPU heterogeneous mode for the device.

When in heterogeneous mode, a vGPU can concurrently host timesliced vGPUs with differing framebuffer sizes.

API would return an appropriate error code upon unsuccessful activation. For example, the heterogeneous mode set will fail with error *NVML_ERROR_IN_USE* if any vGPU instance is active on the device. The caller of this API is expected to shutdown the vGPU VMs and retry setting the *mode*. On KVM platform, setting heterogeneous mode is allowed, if no MDEV device is created on the device, else will fail with same error *NVML_ERROR_IN_USE*. On successful return, the function updates the vGPU heterogeneous mode with the user provided *pHeterogeneousMode->mode*. *pHeterogeneousMode->version* is the version number of the structure nvmlVgpuHeterogeneousMode_t, the caller should set the correct version number to set the vGPU heterogeneous mode.

Parameters

- **device** – Identifier of the target device
- **pHeterogeneousMode** – Pointer to the caller-provided structure of nvmlVgpuHeterogeneousMode_t

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* or *pHeterogeneousMode* is NULL or *pHeterogeneousMode->mode* is invalid

- *NVML_ERROR_IN_USE* If the *device* is in use
- *NVML_ERROR_NO_PERMISSION* If user doesn't have permission to perform the operation
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* doesn't support this feature
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pHeterogeneousMode* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceSetVirtualizationMode**(
nvmlDevice_t device, *nvmlGpuVirtualizationMode_t* virtualMode,
)

This method is used to set the virtualization mode corresponding to the GPU.

For Kepler or newer fully supported devices.

Parameters

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

Returns

- *NVML_SUCCESS* if *virtualMode* is set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid or *virtualMode* is NULL
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_NOT_SUPPORTED* if setting of virtualization mode is not supported.
- *NVML_ERROR_NO_PERMISSION* if setting of virtualization mode is not allowed for this client.

nvmlReturn_t **nvmlDeviceVgpuForceGspUnload**(*nvmlDevice_t* device)

Executes a forced GSP unload operation on a device.

For Ada or newer fully supported devices. Forces the unload of the GSP firmware on a device currently operating in vGPU mode. This operation forcibly removes the GSP from the targeted GPU and terminates all GSP operations.

Note

This is a disruptive operation that will impact any active vGPU instances and should only be used when absolutely necessary, such as during error recovery or maintenance operations.

Warning

This operation may result in a temporary loss of GPU functionality and should be used with caution.

Parameters

device – The identifier of the target device

Returns

- *NVML_SUCCESS* GSP reset completed successfully
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid or null
- *NVML_ERROR_NOT_SUPPORTED* The API is not supported in current state, or *device* not in vGPU mode
- *NVML_ERROR_NO_PERMISSION* The user doesn't have permission to perform this operation
- *NVML_ERROR_UNKNOWN* On any unexpected error during GSP reset operation

nvmlReturn_t **nvmlVgpuInstanceGetPlacementId**(
nvmlVgpuInstance_t vgpuInstance, *nvmlVgpuPlacementId_t* *pPlacement,
)

Query the placement ID of active vGPU instance.

When in vGPU heterogeneous mode, this function returns a valid placement ID as *pPlacement->placementId* else NVML_INVALID_VGPU_PLACEMENT_ID is returned. *pPlacement->version* is the version number of the structure nvmlVgpuPlacementId_t, the caller should set the correct version number to get placement id of the vGPU instance *vgpuInstance*.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **pPlacement** – Pointer to vGPU placement ID structure *nvmlVgpuPlacementId_t*

Returns

- *NVML_SUCCESS* If information is successfully retrieved
- *NVML_ERROR_NOT_FOUND* If *vgpuInstance* does not match a valid active vGPU instance
- *NVML_ERROR_INVALID_ARGUMENT* If *vgpuInstance* is invalid or *pPlacement* is NULL
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pPlacement* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetRuntimeStateSize**(
nvmlVgpuInstance_t vgpuInstance, *nvmlVgpuRuntimeState_t* *pState,
)

Retrieve the currently used runtime state size of the vGPU instance.

This size represents the maximum in-memory data size utilized by a vGPU instance during standard operation. This measurement is exclusive of frame buffer (FB) data size assigned to the vGPU instance.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **pState** – Pointer to the vGPU runtime state's structure *nvmlVgpuRuntimeState_t*

Returns

- *NVML_SUCCESS* If information is successfully retrieved
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *vgpuInstance* is invalid, or *pState* is NULL
- *NVML_ERROR_NOT_FOUND* If *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pState* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetFbReservation**(
    nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *fbReservation,
)

Retrieve the static framebuffer reservation of the vGPU type in bytes.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **fbReservation** – Reference to return the framebuffer reservation

Returns

- *NVML_SUCCESS* Successful completion
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *vgpuTypeId* is invalid, or *fbReservation* is NULL
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetGspHeapSize**(
    nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *gspHeapSize,
)

Retrieve the static GSP heap size of the vGPU type in bytes.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **gspHeapSize** – Reference to return the GSP heap size value

Returns

- *NVML_SUCCESS* Successful completion
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *vgpuTypeId* is invalid, or *gspHeapSize* is NULL
- *NVML_ERROR_UNKNOWN* On any unexpected error

5.2.1.33 vGPU Enums, Constants, Structs

5.2.1.33.1 vGPU Constants

5.2.1.33.1.1 Macros

INVALID_GPU_INSTANCE_ID

INVALID_GPU_INSTANCE_PROFILE_ID

NVML_GRID_LICENSE_BUFFER_SIZE
Buffer size guaranteed to be large enough for *nvmlVgpuTypeGetLicense* .

NVML_GRID_LICENSE_FEATURE_MAX_COUNT

NVML_INVALID_VGPU_PLACEMENT_ID

NVML_VGPU_NAME_BUFFER_SIZE

NVML_VGPU_PGPU_HETEROGENEOUS_MODE
Macros to indicate the vGPU mode of the GPU.

NVML_VGPU_PGPU_HOMOGENEOUS_MODE
Homogeneous vGPU mode.

NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION
Physical GPU migration capability.

NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION_NO
Physical GPU migration is not supported.

NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION_YES
Physical GPU migration is supported.

NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION
vGPU migration capability.

NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION_NO
vGPU migration is not supported.

NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION_YES
vGPU migration is supported.

5.2.1.33.1.2 Macros

INVALID_GPU_INSTANCE_ID

INVALID_GPU_INSTANCE_PROFILE_ID

NVML_GRID_LICENSE_BUFFER_SIZE

Buffer size guaranteed to be large enough for *nvmlVgpuTypeGetLicense*.

NVML_GRID_LICENSE_FEATURE_MAX_COUNT

NVML_INVALID_VGPU_PLACEMENT_ID

NVML_VGPU_NAME_BUFFER_SIZE

NVML_VGPU_PGPU_HETEROGENEOUS_MODE

Macros to indicate the vGPU mode of the GPU.

Heterogeneous vGPU mode.

NVML_VGPU_PGPU_HOMOGENEOUS_MODE

Homogeneous vGPU mode.

NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION

Physical GPU migration capability.

Macros for pGPU's virtualization capabilities bitfield.

NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION_NO

Physical GPU migration is not supported.

NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION_YES

Physical GPU migration is supported.

NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION

vGPU migration capability.

Macros for vGPU instance's virtualization capabilities bitfield.

NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION_NO

vGPU migration is not supported.

NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION_YES

vGPU migration is supported.

5.2.1.33.2 vGPU Enums

5.2.1.33.2.1 Macros

NVML_GRID_LICENSE_EXPIRY_INVALID
Invalid expiry or error fetching expiry.

NVML_GRID_LICENSE_EXPIRY_NOT_APPLICABLE
Expiry not applicable.

NVML_GRID_LICENSE_EXPIRY_NOT_AVAILABLE
Status codes for license expiry.

NVML_GRID_LICENSE_EXPIRY_PERMANENT
Permanent expiry.

NVML_GRID_LICENSE_EXPIRY_VALID
Valid expiry.

5.2.1.33.2.2 Enumerations

nvmlDeviceVgpuCapability_t
Device vGPU queryable capabilities.

nvmlGpuVirtualizationMode_t

nvmlGridLicenseFeatureCode_t
vGPU software licensable features

nvmlHostVgpuMode_t
Host vGPU modes.

nvmlVgpuCapability_t
vGPU queryable capabilities

nvmlVgpuDriverCapability_t
vGPU driver queryable capabilities

nvmlVgpuGuestInfoState_t
vGPU GUEST info state

nvmlVgpuVmIdType_t

5.2.1.33.2.3 Macros

NVML_GRID_LICENSE_EXPIRY_INVALID
Invalid expiry or error fetching expiry.

NVML_GRID_LICENSE_EXPIRY_NOT_APPLICABLE
Expiry not applicable.

NVML_GRID_LICENSE_EXPIRY_NOT_AVAILABLE
Status codes for license expiry.

Expiry information not available

NVML_GRID_LICENSE_EXPIRY_PERMANENT

Permanent expiry.

NVML_GRID_LICENSE_EXPIRY_VALID

Valid expiry.

5.2.1.33.2.4 Enumerations

enum **nvmlDeviceVgpuCapability_t**

Device vGPU queryable capabilities.

Values:

enumerator **NVML_DEVICE_VGPU_CAP_FRACTIONAL_MULTI_VGPU**

Query whether the fractional vGPU profiles on this GPU can be used in multi-vGPU configurations.

enumerator **NVML_DEVICE_VGPU_CAP_HETEROGENEOUS_TIMESLICE_PROFILES**

Query whether the GPU support concurrent execution of timesliced vGPU profiles of differing types.

enumerator **NVML_DEVICE_VGPU_CAP_HETEROGENEOUS_TIMESLICE_SIZES**

Query whether the GPU support concurrent execution of timesliced vGPU profiles of differing framebuffer sizes.

enumerator **NVML_DEVICE_VGPU_CAP_READ_DEVICE_BUFFER_BW**

Query the GPU's read_device_buffer expected bandwidth capacity in megabytes per second.

enumerator **NVML_DEVICE_VGPU_CAP_WRITE_DEVICE_BUFFER_BW**

Query the GPU's write_device_buffer expected bandwidth capacity in megabytes per second.

enumerator **NVML_DEVICE_VGPU_CAP_DEVICE_STREAMING**

Query whether the vGPU profiles on the GPU supports migration data streaming.

enumerator **NVML_DEVICE_VGPU_CAP_MINI_QUARTER_GPU**

Set/Get support for mini-quarter vGPU profiles.

enumerator **NVML_DEVICE_VGPU_CAP_COMPUTE_MEDIA_ENGINE_GPU**

Set/Get support for compute media engine vGPU profiles.

enumerator **NVML_DEVICE_VGPU_CAP_WARM_UPDATE**

Query whether the GPU supports FSR and warm update.

enumerator **NVML_DEVICE_VGPU_CAP_HOMOGENEOUS_PLACEMENTS**

Query whether the GPU supports reporting of placements of timesliced vGPU profiles with identical framebuffer sizes.

enumerator **NVML_DEVICE_VGPU_CAP_MIG_TIMESLICING_SUPPORTED**
Query whether the GPU supports timesliced vGPU on MIG.

enumerator **NVML_DEVICE_VGPU_CAP_MIG_TIMESLICING_ENABLED**
Set/Get MIG timesliced mode reporting, without impacting the underlying functionality.

enumerator **NVML_DEVICE_VGPU_CAP_UVM_STUN**
Query whether the GPU supports UVM stun.

enumerator **NVML_DEVICE_VGPU_CAP_COUNT**

enum **nvmlGpuVirtualizationMode_t**
GPU virtualization mode types.
Values:

enumerator **NVML_GPU_VIRTUALIZATION_MODE_NONE**
Represents Bare Metal GPU.

enumerator **NVML_GPU_VIRTUALIZATION_MODE_PASSTHROUGH**
Device is associated with GPU-Passthorugh.

enumerator **NVML_GPU_VIRTUALIZATION_MODE_VGPU**
Device is associated with vGPU inside virtual machine.

enumerator **NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU**
Device is associated with VGX hypervisor in vGPU mode.

enumerator **NVML_GPU_VIRTUALIZATION_MODE_HOST_VSGA**
Device is associated with VGX hypervisor in vSGA mode.

enum **nvmlGridLicenseFeatureCode_t**
vGPU software licensable features
Values:

enumerator **NVML_GRID_LICENSE_FEATURE_CODE_UNKNOWN**
Unknown.

enumerator **NVML_GRID_LICENSE_FEATURE_CODE_VGPU**
Virtual GPU.

enumerator **NVML_GRID_LICENSE_FEATURE_CODE_NVIDIA_RTX**
Nvidia RTX.

enumerator **NVML_GRID_LICENSE_FEATURE_CODE_VWORKSTATION**
Deprecated, do not use.

enumerator **NVML_GRID_LICENSE_FEATURE_CODE_GAMING**
Gaming.

enumerator **NVML_GRID_LICENSE_FEATURE_CODE_COMPUTE**
Compute.

enumerator **NVML_GRID_LICENSE_FEATURE_CODE_VGAMEDEV**
vGameDev

enum **nvmlHostVgpuMode_t**
Host vGPU modes.
Values:

enumerator **NVML_HOST_VGPU_MODE_NON_SRIOV**
Non SR-IOV mode.

enumerator **NVML_HOST_VGPU_MODE_SRIOV**
SR-IOV mode.

enum **nvmlVgpuCapability_t**
vGPU queryable capabilities
Values:

enumerator **NVML_VGPU_CAP_NVLINK_P2P**
P2P over NVLink is supported.

enumerator **NVML_VGPU_CAP_GPUDIRECT**
GPUDirect capability is supported.

enumerator **NVML_VGPU_CAP_MULTI_VGPU_EXCLUSIVE**
vGPU profile cannot be mixed with other vGPU profiles in same VM

enumerator **NVML_VGPU_CAP_EXCLUSIVE_TYPE**
vGPU profile cannot run on a GPU alongside other profiles of different type

enumerator **NVML_VGPU_CAP_EXCLUSIVE_SIZE**
vGPU profile cannot run on a GPU alongside other profiles of different size

enumerator **NVML_VGPU_CAP_COUNT**

enum **nvmlVgpuDriverCapability_t**
vGPU driver queryable capabilities
Values:

enumerator **NVML_VGPU_DRIVER_CAP_HETEROGENEOUS_MULTI_VGPU**

Supports mixing of different vGPU profiles within one guest VM.

enumerator **NVML_VGPU_DRIVER_CAP_WARM_UPDATE**

Supports FSR and warm update of vGPU host driver without terminating the running guest VM.

enumerator **NVML_VGPU_DRIVER_CAP_COUNT**

enum **nvmlVgpuGuestInfoState_t**

vGPU GUEST info state

Values:

enumerator **NVML_VGPU_INSTANCE_GUEST_INFO_STATE_UNINITIALIZED**

Guest-dependent fields uninitialized.

enumerator **NVML_VGPU_INSTANCE_GUEST_INFO_STATE_INITIALIZED**

Guest-dependent fields initialized.

enum **nvmlVgpuVmIdType_t**

Types of VM identifiers

Values:

enumerator **NVML_VGPU_VM_ID_DOMAIN_ID**

VM ID represents DOMAIN ID.

enumerator **NVML_VGPU_VM_ID_UUID**

VM ID represents UUID.

5.2.1.33.3 vGPU Structs

5.2.1.33.3.1 Macros

NVML_GRID_LICENSE_STATE_LICENSED

Licensed state.

NVML_GRID_LICENSE_STATE_UNINITIALIZED

Uninitialized state.

NVML_GRID_LICENSE_STATE_UNKNOWN

vGPU license state

NVML_GRID_LICENSE_STATE_UNLICENSED

Unlicensed state.

NVML_GRID_LICENSE_STATE_UNLICENSED_RESTRICTED

Unlicensed restricted state.

NVML_GRID_LICENSE_STATE_UNLICENSED_UNRESTRICTED

Unlicensed unrestricted state.

NVML_SCHEDULER_SW_MAX_LOG_ENTRIES
Maximum number of scheduler log entries.

NVML_SUPPORTED_VGPU_SCHEDULER_POLICY_COUNT
Number of supported vGPU scheduler policies.

NVML_VGPU_SCHEDULER_ARR_DEFAULT
Default Adaptive Round Robin mode.

NVML_VGPU_SCHEDULER_ARR_DISABLE
Disable Adaptive Round Robin mode.

NVML_VGPU_SCHEDULER_ARR_ENABLE
Enable Adaptive Round Robin mode.

NVML_VGPU_SCHEDULER_ENGINE_TYPE_GRAPHICS
vGPU scheduler engine types A GPU or GI may support a subset of engines

NVML_VGPU_SCHEDULER_ENGINE_TYPE_NVENC0
NVENC0.

NVML_VGPU_SCHEDULER_ENGINE_TYPE_NVENC1
NVENC1.

NVML_VGPU_SCHEDULER_POLICY_BEST_EFFORT
Best effort scheduler policy.

NVML_VGPU_SCHEDULER_POLICY_EQUAL_SHARE
Equal share scheduler policy.

NVML_VGPU_SCHEDULER_POLICY_FIXED_SHARE
Fixed share scheduler policy.

NVML_VGPU_SCHEDULER_POLICY_UNKNOWN
vGPU scheduler policies

nvmlActiveVgpuInstanceInfo_v1
Version macro for *nvmlActiveVgpuInstanceInfo_v1_t* .

nvmlVgpuCreatablePlacementInfo_v1
Version macro for *nvmlVgpuCreatablePlacementInfo_v1_t* .

nvmlVgpuHeterogeneousMode_v1
Version macro for *nvmlVgpuHeterogeneousMode_v1_t* .

nvmlVgpuInstancesUtilizationInfo_v1
Version macro for *nvmlVgpuInstancesUtilizationInfo_v1_t* .

nvmlVgpuPlacementId_v1
Version macro for *nvmlVgpuPlacementId_v1_t* .

nvmlVgpuPlacementList_v1
Version macro for *nvmlVgpuPlacementList_v1_t* .

nvmlVgpuPlacementList_v2
Version macro for *nvmlVgpuPlacementList_v2_t* .

nvmlVgpuProcessesUtilizationInfo_v1
Version macro for *nvmlVgpuProcessesUtilizationInfo_v1_t* .

nvmlVgpuRuntimeState_v1
Version macro for *nvmlVgpuRuntimeState_v1_t* .

nvmlVgpuSchedulerLogInfo_v1
Version macro for *nvmlVgpuSchedulerLogInfo_v1_t* .

nvmlVgpuSchedulerStateInfo_v1
Version macro for *nvmlVgpuSchedulerStateInfo_v1_t* .

nvmlVgpuSchedulerState_v1
Version macro for *nvmlVgpuSchedulerState_v1_t* .

nvmlVgpuTypeBar1Info_v1
Version macro for *nvmlVgpuTypeBar1Info_v1_t* .

nvmlVgpuTypeIdInfo_v1
Version macro for *nvmlVgpuTypeIdInfo_v1_t* .

nvmlVgpuTypeMaxInstance_v1
Version macro for *nvmlVgpuTypeMaxInstance_v1_t* .

5.2.1.33.3.2 Enumerations

nvmlDeviceGpuRecoveryAction_t
Enum describing the GPU Recovery Action.

5.2.1.33.3.3 Structs

nvmlActiveVgpuInstanceInfo_v1_t
Structure to store active vGPU instance information Version 1.

nvmlGridLicensableFeature_t
Structure containing vGPU software licensable feature information.

nvmlGridLicensableFeatures_t
Structure to store vGPU software licensable features.

nvmlGridLicenseExpiry_t
Structure to store license expiry date and time values.

nvmlVgpuCreatablePlacementInfo_v1_t
Structure to store creatable vGPU placement information version 1.

nvmlVgpuHeterogeneousMode_v1_t
Structure to store the vGPU heterogeneous mode of device version 1.

nvmlVgpuInstanceUtilizationInfo_v1_t
Structure to store Utilization Value and vgpuInstance Info Version 1.

nvmlVgpuInstanceUtilizationSample_t
Structure to store Utilization Value and vgpuInstance.

nvmlVgpuInstancesUtilizationInfo_v1_t
Structure to store recent utilization for vGPU instances running on a device version 1.

nvmlVgpuLicenseExpiry_t
Structure to store the vGPU license expiry details.

nvmlVgpuLicenseInfo_t

nvmlVgpuPlacementId_v1_t
Structure to store the placement ID of vGPU instance version 1.

nvmlVgpuPlacementList_v1_t
Structure to store the list of vGPU placements version 1.

nvmlVgpuPlacementList_v2_t
Structure to store the list of vGPU placements version 2.

nvmlVgpuProcessUtilizationInfo_v1_t
Structure to store Utilization Value, vgpuInstance and subprocess information for process running on vGPU instance version 1.

nvmlVgpuProcessUtilizationSample_t
Structure to store Utilization Value, vgpuInstance and subprocess information.

nvmlVgpuProcessesUtilizationInfo_v1_t
Structure to store recent utilization, vgpuInstance and subprocess information for processes running on vGPU instances active on a device version 1.

nvmlVgpuRuntimeState_v1_t
Structure to store the information of vGPU runtime state version 1.

nvmlVgpuSchedulerCapabilities_t
Structure to store the vGPU scheduler capabilities.

nvmlVgpuSchedulerGetState_t
Structure to store the vGPU scheduler state.

nvmlVgpuSchedulerLogEntry_t
Structure to store the state and logs of a software runlist.

nvmlVgpuSchedulerLogEntry_v2_t
Structure to store the state and logs of a software runlist.

nvmlVgpuSchedulerLogInfo_v1_t
Structure to store vGPU scheduler log information Version 1.

nvmlVgpuSchedulerLogInfo_v2_t
Structure to store vGPU scheduler log information.

nvmlVgpuSchedulerLog_t
Structure to store a vGPU software scheduler log.

nvmlVgpuSchedulerSetState_t
Structure to set the vGPU scheduler state.

nvmlVgpuSchedulerStateInfo_v1_t
Structure to store vGPU scheduler state information Version 1.

nvmlVgpuSchedulerStateInfo_v2_t
Structure to store vGPU scheduler state information.

nvmlVgpuSchedulerState_v1_t
Structure to set vGPU scheduler state information version 1.

nvmlVgpuSchedulerState_v2_t
Structure to set vGPU scheduler state information.

nvmlVgpuTypeBar1Info_v1_t
Structure to store BAR1 size information of vGPU type Version 1.

nvmlVgpuTypeIdInfo_v1_t
Structure to store the vGPU type IDs version 1.

nvmlVgpuTypeMaxInstance_v1_t
Structure to store the maximum number of possible vGPU type IDs version 1.

5.2.1.33.3.4 Typedefs

nvmlActiveVgpuInstanceInfo_t

nvmlVgpuCreatablePlacementInfo_t

nvmlVgpuHeterogeneousMode_t

nvmlVgpuInstance_t

nvmlVgpuInstancesUtilizationInfo_t

nvmlVgpuPlacementId_t

nvmlVgpuPlacementList_t

nvmlVgpuProcessesUtilizationInfo_t

nvmlVgpuRuntimeState_t

nvmlVgpuSchedulerLogInfo_t

nvmlVgpuSchedulerStateInfo_t

nvmlVgpuSchedulerState_t

nvmlVgpuTypeBar1Info_t

nvmlVgpuTypeIdInfo_t

nvmlVgpuTypeId_t

nvmlVgpuTypeMaxInstance_t

5.2.1.33.3.5 Unions

nvmlVgpuSchedulerParams_t
: Union to represent the vGPU Scheduler Parameters.

nvmlVgpuSchedulerSetParams_t
: Union to represent the vGPU Scheduler set Parameters.

5.2.1.33.3.6 Macros

NVML_GRID_LICENSE_STATE_LICENSED

Licensed state.

NVML_GRID_LICENSE_STATE_UNINITIALIZED

Uninitialized state.

NVML_GRID_LICENSE_STATE_UNKNOWN

vGPU license state

Unknown state

NVML_GRID_LICENSE_STATE_UNLICENSED

Unlicensed state.

NVML_GRID_LICENSE_STATE_UNLICENSED_RESTRICTED

Unlicensed restricted state.

NVML_GRID_LICENSE_STATE_UNLICENSED_UNRESTRICTED

Unlicensed unrestricted state.

NVML_SCHEDULER_SW_MAX_LOG_ENTRIES

Maximum number of scheduler log entries.

NVML_SUPPORTED_VGPU_SCHEDULER_POLICY_COUNT

Number of supported vGPU scheduler policies.

NVML_VGPU_SCHEDULER_ARR_DEFAULT

Default Adaptive Round Robin mode.

NVML_VGPU_SCHEDULER_ARR_DISABLE

Disable Adaptive Round Robin mode.

NVML_VGPU_SCHEDULER_ARR_ENABLE

Enable Adaptive Round Robin mode.

NVML_VGPU_SCHEDULER_ENGINE_TYPE_GRAPHICS

vGPU scheduler engine types A GPU or GI may support a subset of engines

Graphics engine.

NVML_VGPU_SCHEDULER_ENGINE_TYPE_NVENC0

NVENC0.

NVML_VGPU_SCHEDULER_ENGINE_TYPE_NVENC1

NVENC1.

NVML_VGPU_SCHEDULER_POLICY_BEST_EFFORT
Best effort scheduler policy.

NVML_VGPU_SCHEDULER_POLICY_EQUAL_SHARE
Equal share scheduler policy.

NVML_VGPU_SCHEDULER_POLICY_FIXED_SHARE
Fixed share scheduler policy.

NVML_VGPU_SCHEDULER_POLICY_UNKNOWN
vGPU scheduler policies

Unknown scheduler policy.

nvmlActiveVgpuInstanceInfo_v1
Version macro for nvmlActiveVgpuInstanceInfo_v1_t.

nvmlVgpuCreatablePlacementInfo_v1
Version macro for nvmlVgpuCreatablePlacementInfo_v1_t.

nvmlVgpuHeterogeneousMode_v1
Version macro for nvmlVgpuHeterogeneousMode_v1_t.

nvmlVgpuInstancesUtilizationInfo_v1
Version macro for nvmlVgpuInstancesUtilizationInfo_v1_t.

nvmlVgpuPlacementId_v1
Version macro for nvmlVgpuPlacementId_v1_t.

nvmlVgpuPlacementList_v1
Version macro for nvmlVgpuPlacementList_v1_t.

nvmlVgpuPlacementList_v2
Version macro for nvmlVgpuPlacementList_v2_t.

nvmlVgpuProcessesUtilizationInfo_v1
Version macro for nvmlVgpuProcessesUtilizationInfo_v1_t.

nvmlVgpuRuntimeState_v1
Version macro for nvmlVgpuRuntimeState_v1_t.

nvmlVgpuSchedulerLogInfo_v1
Version macro for nvmlVgpuSchedulerLogInfo_v1_t.

nvmlVgpuSchedulerStateInfo_v1
Version macro for nvmlVgpuSchedulerStateInfo_v1_t.

nvmlVgpuSchedulerState_v1
Version macro for nvmlVgpuSchedulerState_v1_t.

nvmlVgpuTypeBar1Info_v1
Version macro for nvmlVgpuTypeBar1Info_v1_t.

nvmlVgpuTypeIdInfo_v1
Version macro for nvmlVgpuTypeIdInfo_v1_t.

nvmlVgpuTypeMaxInstance_v1
Version macro for nvmlVgpuTypeMaxInstance_v1_t.

5.2.1.33.3.7 Enumerations

enum **nvmlDeviceGpuRecoveryAction_t**
Enum describing the GPU Recovery Action.
Values:

enumerator **NVML_GPU_RECOVERY_ACTION_NONE**
No action needed.

enumerator **NVML_GPU_RECOVERY_ACTION_GPU_RESET**
Reset Gpu.

enumerator **NVML_GPU_RECOVERY_ACTION_NODE_REBOOT**
Reboot Node.

enumerator **NVML_GPU_RECOVERY_ACTION_DRAIN_P2P**
Drain P2P.

enumerator **NVML_GPU_RECOVERY_ACTION_DRAIN_AND_RESET**
Drain P2P and Reset Gpu.

enumerator **NVML_GPU_RECOVERY_ACTION_RECOVER_IMEX_DOMAIN**
Recover IMEX Domain.

enumerator **NVML_GPU_RECOVERY_ACTION_BUS_RESET**
Reset the GPU's PCIe bus.

enumerator **NVML_GPU_RECOVERY_ACTION_SYSTEM_REBOOT**
Reboot the system.

5.2.1.33.3.8 Typedefs

typedef *nvmlActiveVgpuInstanceInfo_v1_t* **nvmlActiveVgpuInstanceInfo_t**

typedef *nvmlVgpuCreatablePlacementInfo_v1_t* **nvmlVgpuCreatablePlacementInfo_t**

typedef *nvmlVgpuHeterogeneousMode_v1_t* **nvmlVgpuHeterogeneousMode_t**

typedef unsigned int **nvmlVgpuInstance_t**

typedef *nvmlVgpuInstancesUtilizationInfo_v1_t* **nvmlVgpuInstancesUtilizationInfo_t**

typedef *nvmlVgpuPlacementId_v1_t* **nvmlVgpuPlacementId_t**

typedef *nvmlVgpuPlacementList_v2_t* **nvmlVgpuPlacementList_t**

typedef *nvmlVgpuProcessesUtilizationInfo_v1_t* **nvmlVgpuProcessesUtilizationInfo_t**

typedef *nvmlVgpuRuntimeState_v1_t* **nvmlVgpuRuntimeState_t**

typedef *nvmlVgpuSchedulerLogInfo_v1_t* **nvmlVgpuSchedulerLogInfo_t**

typedef *nvmlVgpuSchedulerStateInfo_v1_t* **nvmlVgpuSchedulerStateInfo_t**

typedef *nvmlVgpuSchedulerState_v1_t* **nvmlVgpuSchedulerState_t**

typedef *nvmlVgpuTypeBar1Info_v1_t* **nvmlVgpuTypeBar1Info_t**

typedef *nvmlVgpuTypeIdInfo_v1_t* **nvmlVgpuTypeIdInfo_t**

typedef unsigned int **nvmlVgpuTypeId_t**

typedef *nvmlVgpuTypeMaxInstance_v1_t* **nvmlVgpuTypeMaxInstance_t**

5.2.1.33.4 Groups

vGPU Constants

vGPU Enums

vGPU Structs

5.2.1.34 vGPU Management

This chapter describes APIs supporting NVIDIA vGPU.

5.2.1.34.1 Functions

nvmlReturn_t *nvmlDeviceGetActiveVgpus*(nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuInstance_t *vgpuInstances)
Retrieve the active vGPU instances on a device.

nvmlReturn_t *nvmlDeviceGetCreatableVgpus*(nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuTypeId_t *vgpuTypeIds)
Retrieve the currently creatable vGPU types on a physical GPU (device).

nvmlReturn_t *nvmlDeviceGetSupportedVgpus*(nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuTypeId_t *vgpuTypeIds)
Retrieve the supported vGPU types on a physical GPU (device).

nvmlReturn_t *nvmlDeviceGetVgpuCapabilities*(nvmlDevice_t device, nvmlDeviceVgpuCapability_t capability, unsigned int *capResult)
Retrieve the requested vGPU capability for GPU.

nvmlReturn_t *nvmlDeviceGetVgpuSchedulerLog_v2*(nvmlDevice_t device, nvmlVgpuSchedulerLogInfo_v2_t *pSchedulerLogInfo)
Returns the vGPU Software scheduler logs for the device.

nvmlReturn_t *nvmlDeviceGetVgpuSchedulerState_v2*(nvmlDevice_t device, nvmlVgpuSchedulerStateInfo_v2_t *pSchedulerStateInfo)
Returns the vGPU scheduler state.

nvmlReturn_t *nvmlDeviceSetVgpuSchedulerState_v2*(nvmlDevice_t device, nvmlVgpuSchedulerState_v2_t *pSchedulerState)
Sets the vGPU scheduler state.

nvmlReturn_t *nvmlGetVgpuDriverCapabilities*(nvmlVgpuDriverCapability_t capability, unsigned int *capResult)
Retrieve the requested vGPU driver capability.

nvmlReturn_t *nvmlGpuInstanceGetActiveVgpus*(nvmlGpuInstance_t gpuInstance, nvmlActiveVgpuInstanceInfo_t *pVgpuInstanceInfo)
Retrieve the active vGPU instances within a GPU instance.

nvmlReturn_t *nvmlGpuInstanceGetCreatableVgpus*(nvmlGpuInstance_t gpuInstance, nvmlVgpuTypeIdInfo_t *pVgpus)
Query the currently creatable vGPU types on a specific GPU Instance.

nvmlReturn_t *nvmlGpuInstanceGetVgpuHeterogeneousMode*(nvmlGpuInstance_t gpuInstance, nvmlVgpuHeterogeneousMode_t *pHeterogeneousMode)
Get the vGPU heterogeneous mode for the GPU instance.

nvmlReturn_t *nvmlGpuInstanceGetVgpuSchedulerLog*(nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerLogInfo_t *pSchedulerLogInfo)

nvmlReturn_t *nvmlGpuInstanceGetVgpuSchedulerLog_v2*(nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerLogInfo_v2_t *pSchedulerLogInfo)
Returns the vGPU scheduler logs for the given GPU instance.

nvmlReturn_t *nvmlGpuInstanceGetVgpuSchedulerState*(nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerStateInfo_t *pSchedulerStateInfo)

nvmlReturn_t *nvmlGpuInstanceGetVgpuSchedulerState_v2*(nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerStateInfo_v2_t *pSchedulerStateInfo)
Returns the vGPU scheduler state for the given GPU instance.

nvmlReturn_t *nvmlGpuInstanceGetVgpuTypeCreatablePlacements*(nvmlGpuInstance_t gpuInstance, nvmlVgpuCreatablePlacementInfo_t *pCreatablePlacementInfo)
Query the creatable vGPU placement ID of the vGPU type within a GPU instance.

nvmlReturn_t *nvmlGpuInstanceSetVgpuHeterogeneousMode*(nvmlGpuInstance_t gpuInstance, const nvmlVgpuHeterogeneousMode_t *pHeterogeneousMode)
Enable or disable vGPU heterogeneous mode for the GPU instance.

nvmlReturn_t *nvmlGpuInstanceSetVgpuSchedulerState*(nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerState_t *pScheduler)

nvmlReturn_t *nvmlGpuInstanceSetVgpuSchedulerState_v2*(nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerState_v2_t *pSchedulerState)
Set vGPU scheduler state for the given GPU instance.

nvmlReturn_t *nvmlVgpuInstanceGetEccMode*(nvmlVgpuInstance_t vgpuInstance, nvmlEnableState_t *eccMode)
Retrieve the current ECC mode of vGPU instance.

nvmlReturn_t *nvmlVgpuInstanceGetEncoderCapacity*(nvmlVgpuInstance_t vgpuInstance, unsigned int *encoderCapacity)
Retrieve the encoder capacity of a vGPU instance, as a percentage of maximum encoder capacity with valid values in the range 0-100.

nvmlReturn_t *nvmlVgpuInstanceGetEncoderSessions*(nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, nvmlEncoderSessionInfo_t *sessionInfo)
Retrieves information about all active encoder sessions on a vGPU Instance.

nvmlReturn_t *nvmlVgpuInstanceGetEncoderStats*(nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, unsigned int *averageFps, unsigned int *averageLatency)
Retrieves the current encoder statistics of a vGPU Instance.

nvmlReturn_t *nvmlVgpuInstanceGetFBCSessions*(nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, nvmlFBCSessionInfo_t *sessionInfo)
Retrieves information about active frame buffer capture sessions on a vGPU Instance.

nvmlReturn_t *nvmlVgpuInstanceGetFBCStats*(nvmlVgpuInstance_t vgpuInstance, nvmlFBCStats_t *fbcStats)
Retrieves the active frame buffer capture sessions statistics of a vGPU Instance.

nvmlReturn_t *nvmlVgpuInstanceGetFbUsage*(nvmlVgpuInstance_t vgpuInstance, unsigned long long *fbUsage)
Retrieve the framebuffer usage in bytes.

nvmlReturn_t *nvmlVgpuInstanceGetFrameRateLimit*(nvmlVgpuInstance_t vgpuInstance, unsigned int *frameRateLimit)
Retrieve the frame rate limit set for the vGPU instance.

nvmlReturn_t *nvmlVgpuInstanceGetGpuInstanceId*(nvmlVgpuInstance_t vgpuInstance, unsigned int *gpuInstanceId)
Retrieve the GPU Instance ID for the given vGPU Instance.

nvmlReturn_t *nvmlVgpuInstanceGetGpuPciId*(nvmlVgpuInstance_t vgpuInstance, char *vgpuPciId, unsigned int *length)
Retrieves the PCI Id of the given vGPU Instance i.e.

nvmlReturn_t *nvmlVgpuInstanceGetLicenseStatus*(nvmlVgpuInstance_t vgpuInstance, unsigned int *licensed)

nvmlReturn_t *nvmlVgpuInstanceGetMdevUUID*(nvmlVgpuInstance_t vgpuInstance, char *mdevUuid, unsigned int size)
Retrieve the MDEV UUID of a vGPU instance.

nvmlReturn_t *nvmlVgpuInstanceGetType*(nvmlVgpuInstance_t vgpuInstance, nvmlVgpuTypeId_t *vgpuTypeId)
Retrieve the vGPU type of a vGPU instance.

nvmlReturn_t *nvmlVgpuInstanceGetUUID*(nvmlVgpuInstance_t vgpuInstance, char *uuid, unsigned int size)
Retrieve the UUID of a vGPU instance.

nvmlReturn_t *nvmlVgpuInstanceGetVmDriverVersion*(nvmlVgpuInstance_t vgpuInstance, char *version, unsigned int length)
Retrieve the NVIDIA driver version installed in the VM associated with a vGPU.

nvmlReturn_t *nvmlVgpuInstanceGetVmID*(nvmlVgpuInstance_t vgpuInstance, char *vmId, unsigned int size, nvmlVgpuVmIdType_t *vmIdType)
Retrieve the VM ID associated with a vGPU instance.

nvmlReturn_t *nvmlVgpuInstanceSetEncoderCapacity*(nvmlVgpuInstance_t vgpuInstance, unsigned int encoderCapacity)
Set the encoder capacity of a vGPU instance, as a percentage of maximum encoder capacity with valid values in the range 0-100.

nvmlReturn_t *nvmlVgpuTypeGetBAR1Info*(nvmlVgpuTypeId_t vgpuTypeId, nvmlVgpuTypeBar1Info_t *bar1Info)
Retrieve the BAR1 info for given vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetCapabilities*(nvmlVgpuTypeId_t vgpuTypeId, nvmlVgpuCapability_t capability, unsigned int *capResult)
Retrieve the requested capability for a given vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetClass*(nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeClass, unsigned int *size)
Retrieve the class of a vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetDeviceID*(nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *deviceID, unsigned long long *subsystemID)
Retrieve the device ID of a vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetFrameRateLimit*(nvmlVgpuTypeId_t vgpuTypeId, unsigned int *frameRateLimit)
Retrieve the static frame rate limit value of the vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetFramebufferSize*(nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *fbSize)
Retrieve the vGPU framebuffer size in bytes.

nvmlReturn_t *nvmlVgpuTypeGetGpuInstanceProfileId*(nvmlVgpuTypeId_t vgpuTypeId, unsigned int *gpuInstanceProfileId)
Retrieve the GPU Instance Profile ID for the given vGPU type ID.

nvmlReturn_t *nvmlVgpuTypeGetLicense*(nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeLicenseString, unsigned int size)
Retrieve license requirements for a vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetMaxInstances*(nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCount)
Retrieve the maximum number of vGPU instances creatable on a device for given vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetMaxInstancesPerGpuInstance*(nvmlVgpuTypeMaxInstance_t *pMaxInstance)
Retrieve the maximum number of vGPU instances per GPU instance for given vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetMaxInstancesPerVm*(nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCountPerVm)
Retrieve the maximum number of vGPU instances supported per VM for given vGPU type.

nvmlReturn_t *nvmlVgpuTypeGetName*(nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeName, unsigned int *size)
Retrieve the vGPU type name.

nvmlReturn_t *nvmlVgpuTypeGetNumDisplayHeads*(nvmlVgpuTypeId_t vgpuTypeId, unsigned int *numDisplayHeads)
Retrieve count of vGPU's supported display heads.

nvmlReturn_t *nvmlVgpuTypeGetResolution*(nvmlVgpuTypeId_t vgpuTypeId, unsigned int displayIndex, unsigned int *xdim, unsigned int *ydim)
Retrieve vGPU display head's maximum supported resolution.

5.2.1.34.2 Functions

nvmlReturn_t **nvmlDeviceGetActiveVgpus**(
nvmlDevice_t device, unsigned int *vgpuCount, *nvmlVgpuInstance_t* *vgpuInstances,
)

Retrieve the active vGPU instances on a device.

An array of active vGPU instances is returned in the caller-supplied buffer pointed at by *vgpuInstances*. The array element count is passed in *vgpuCount*, and *vgpuCount* is used to return the number of vGPU instances written to the buffer.

If the supplied buffer is not large enough to accommodate the vGPU instance array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlVgpuInstance_t array required in *vgpuCount*. To query the number of active vGPU instances, call this function with *vgpuCount = 0. The code will return NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if no vGPU Types are supported.

For Kepler or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **vgpuCount** – Pointer which passes in the array size as well as get back the number of types
- **vgpuInstances** – Pointer to array in which to return list of vGPU instances

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, or *vgpuCount* is NULL
- *NVML_ERROR_INSUFFICIENT_SIZE* if *size* is too small
- *NVML_ERROR_NOT_SUPPORTED* if vGPU is not supported by the device
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetCreatableVgpus**(
nvmlDevice_t device, unsigned int *vgpuCount, *nvmlVgpuTypeId_t* *vgpuTypeIds,
)

Retrieve the currently creatable vGPU types on a physical GPU (device).

An array of creatable vGPU types for the physical GPU indicated by *device* is returned in the caller-supplied buffer pointed at by *vgpuTypeIds*. The element count of nvmlVgpuTypeId_t array is passed in *vgpuCount*, and *vgpuCount* is used to return the number of vGPU types written to the buffer.

The creatable vGPU types for a device may differ over time, as there may be restrictions on what type of vGPU types can concurrently run on a device. For example, if only one vGPU type is allowed at a time on a device, then the creatable list will be restricted to whatever vGPU type is already running on the device.

If the supplied buffer is not large enough to accommodate the vGPU type array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlVgpuTypeId_t array required in *vgpuCount*. To query the number of vGPU types that can be created for the GPU, call this function with *vgpuCount = 0. The code will return NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if no vGPU types are creatable.

Parameters

- **device** – The identifier of the target device
- **vgpuCount** – Pointer to caller-supplied array size, and returns number of vGPU types
- **vgpuTypeIds** – Pointer to caller-supplied array in which to return list of vGPU types

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_INSUFFICIENT_SIZE* *vgpuTypeIds* buffer is too small, array element count is returned in *vgpuCount*
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuCount* is NULL
- *NVML_ERROR_NOT_SUPPORTED* if vGPU is not supported by the device
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetSupportedVgpus**(
    nvmlDevice_t device, unsigned int *vgpuCount, *nvmlVgpuTypeId_t* *vgpuTypeIds,
)

Retrieve the supported vGPU types on a physical GPU (device).

An array of supported vGPU types for the physical GPU indicated by *device* is returned in the caller-supplied buffer pointed at by *vgpuTypeIds*. The element count of nvmlVgpuTypeId_t array is passed in *vgpuCount*, and *vgpuCount* is used to return the number of vGPU types written to the buffer.

If the supplied buffer is not large enough to accommodate the vGPU type array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlVgpuTypeId_t array required in *vgpuCount*. To query the number of vGPU types supported for the GPU, call this function with *vgpuCount = 0. The code will return NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if no vGPU types are supported.

Parameters

- **device** – The identifier of the target device
- **vgpuCount** – Pointer to caller-supplied array size, and returns number of vGPU types
- **vgpuTypeIds** – Pointer to caller-supplied array in which to return list of vGPU types

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_INSUFFICIENT_SIZE* *vgpuTypeIds* buffer is too small, array element count is returned in *vgpuCount*
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuCount* is NULL or *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* if vGPU is not supported by the device
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuCapabilities**(
    nvmlDevice_t device, *nvmlDeviceVgpuCapability_t* capability, unsigned int *capResult,
)

Retrieve the requested vGPU capability for GPU.

Refer to the *nvmlDeviceVgpuCapability_t* structure for the specific capabilities that can be queried. The return value in *capResult* reports a non-zero value indicating that the capability is supported, and also reports the capability's data based on the queried capability.

For Maxwell or newer fully supported devices.

Parameters

- **device** – The identifier of the target device
- **capability** – Specifies the *nvmlDeviceVgpuCapability_t* to be queried
- **capResult** – Specifies that the queried capability is supported, and also returns capability's data

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, or *capability* is invalid, or *capResult* is NULL
- *NVML_ERROR_NOT_SUPPORTED* the API is not supported in current state or *device* not in vGPU mode
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuSchedulerLog_v2**(
    nvmlDevice_t device, *nvmlVgpuSchedulerLogInfo_v2_t* *pSchedulerLogInfo,
)

Returns the vGPU Software scheduler logs for the device.

pSchedulerLogInfo points to a caller-allocated structure to contain the logs. The number of elements returned will never exceed *NVML_SCHEDULER_SW_MAX_LOG_ENTRIES*.

To get the entire logs, call the function atleast 5 times a second.

For Pascal or newer fully supported devices.

Parameters

- **device** – The identifier of the target *device*
- **pSchedulerLogInfo** – Reference in which *pSchedulerLogInfo* is written

Returns

- *NVML_SUCCESS* vGPU scheduler logs were successfully obtained
- *NVML_ERROR_INVALID_ARGUMENT* If *pSchedulerLogInfo* is NULL or *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* not in vGPU host mode
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuSchedulerState_v2**(
    nvmlDevice_t device, *nvmlVgpuSchedulerStateInfo_v2_t* *pSchedulerStateInfo,
)

Returns the vGPU scheduler state.

The information returned in nvmlVgpuSchedulerStateInfo_v2_t is not relevant if the BEST EFFORT policy is set.

For Pascal or newer fully supported devices.

Parameters

- **device** – The identifier of the target *device*
- **pSchedulerStateInfo** – Reference in which *pSchedulerStateInfo* is returned

Returns

- *NVML_SUCCESS* vGPU scheduler state is successfully obtained
- *NVML_ERROR_INVALID_ARGUMENT* If *pSchedulerStateInfo* is NULL or *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* not in vGPU host mode
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceSetVgpuSchedulerState_v2**(
    nvmlDevice_t device, *nvmlVgpuSchedulerState_v2_t* *pSchedulerState,
)

Sets the vGPU scheduler state.

For Pascal or newer fully supported devices.

The scheduler state change won't persist across module load/unload. Scheduler state and params will be allowed to set only when no VM is running.

Parameters

- **device** – The identifier of the target *device*
- **pSchedulerState** – vGPU *pSchedulerState* to set

Returns

- *NVML_SUCCESS* vGPU scheduler state has been successfully set
- *NVML_ERROR_INVALID_ARGUMENT* If *pSchedulerState* is NULL or *device* is invalid
- *NVML_ERROR_RESET_REQUIRED* If setting *pSchedulerState* failed with fatal error, reboot is required to overcome from this error.
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* not in vGPU host mode or if any vGPU instance currently exists on the *device*
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGetVgpuDriverCapabilities**(
    nvmlVgpuDriverCapability_t capability, unsigned int *capResult,
)

Retrieve the requested vGPU driver capability.

Refer to the *nvmlVgpuDriverCapability_t* structure for the specific capabilities that can be queried. The return value in *capResult* should be treated as a boolean, with a non-zero value indicating that the capability is supported.

For Maxwell or newer fully supported devices.

Parameters

- **capability** – Specifies the *nvmlVgpuDriverCapability_t* to be queried
- **capResult** – A boolean for the queried capability indicating that feature is supported

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *capability* is invalid, or *capResult* is NULL
- *NVML_ERROR_NOT_SUPPORTED* the API is not supported in current state or *devices* not in vGPU mode
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlGpuInstanceGetActiveVgpus**(
    nvmlGpuInstance_t gpuInstance, *nvmlActiveVgpuInstanceInfo_t* *pVgpuInstanceInfo,
)

Retrieve the active vGPU instances within a GPU instance.

An array of active vGPU instances is returned in the caller-supplied buffer pointed at by *pVgpuInstanceInfo->vgpuInstances*. The array element count is passed in *pVgpuInstanceInfo->vgpuCount*, and *pVgpuInstanceInfo->vgpuCount* is used to return the number of vGPU instances written to the buffer.

If the supplied buffer is not large enough to accommodate the vGPU instance array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of nvmlVgpuInstance_t array required in *pVgpuInstanceInfo->vgpuCount*. To query the number of active vGPU instances, call this function with pVgpuInstanceInfo->vgpuCount = 0 and pVgpuInstanceInfo->vgpuTypeIds = NULL. The code will return NVML_ERROR_INSUFFICIENT_SIZE, or NVML_SUCCESS if no vGPU Types are active.

Parameters

- **gpuInstance** – The GPU instance handle
- **pVgpuInstanceInfo** – Pointer to the vGPU instance information structure *nvmlActiveVgpuInstanceInfo_t*

Returns

- *NVML_SUCCESS* Successful completion
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pVgpuInstanceInfo* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_INSUFFICIENT_SIZE* *pVgpuInstanceInfo->vgpuTypeIds* buffer is too small, array element count is returned in *pVgpuInstanceInfo->vgpuCount*
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pVgpuInstanceInfo* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGpuInstanceGetCreatableVgpus**(
    nvmlGpuInstance_t gpuInstance, *nvmlVgpuTypeIdInfo_t* *pVgpus,
)

Query the currently creatable vGPU types on a specific GPU Instance.

The function returns an array of vGPU types that can be created for a specified GPU instance. This array is stored in a caller-supplied buffer, with the buffer's element count passed through *pVgpus->vgpuCount*. The number of vGPU types written to the buffer is indicated by *pVgpus->vgpuCount*. If the buffer is too small to hold the vGPU type array, the function returns NVML_ERROR_INSUFFICIENT_SIZE and updates *pVgpus->vgpuCount* with the required element count.

To determine the creatable vGPUs for a GPU Instance, invoke this function with *pVgpus->vgpuCount* set to 0 and *pVgpus->vgpuTypeIds* as NULL. This will result in NVML_ERROR_INSUFFICIENT_SIZE being returned, along with the count value in *pVgpus->vgpuCount*.

The creatable vGPU types may differ over time, as there may be restrictions on what type of vGPUs can concurrently run on the device.

Parameters

- **gpuInstance** – The GPU instance handle
- **pVgpus** – Pointer to the caller-provided structure of nvmlVgpuTypeIdInfo_t

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pVgpus* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU
- *NVML_ERROR_INSUFFICIENT_SIZE* If *pVgpus->vgpuTypeIds* buffer is small
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pVgpus* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGpuInstanceGetVgpuHeterogeneousMode**(
nvmlGpuInstance_t gpuInstance, *nvmlVgpuHeterogeneousMode_t* *pHeterogeneousMode,
)

Get the vGPU heterogeneous mode for the GPU instance.

When in heterogeneous mode, a vGPU can concurrently host timesliced vGPUs with differing framebuffer sizes.

On successful return, the function returns *pHeterogeneousMode->mode* with the current vGPU heterogeneous mode. *pHeterogeneousMode->version* is the version number of the structure nvmlVgpuHeterogeneousMode_t, the caller should set the correct version number to retrieve the vGPU heterogeneous mode. *pHeterogeneousMode->mode* can either be *NVML_FEATURE_ENABLED* or *NVML_FEATURE_DISABLED*.

For Blackwell &tm GB20x; or newer fully supported devices.

Parameters

- **gpuInstance** – The GPU instance handle
- **pHeterogeneousMode** – Pointer to the caller-provided structure of nvmlVgpuHeterogeneousMode_t

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pHeterogeneousMode* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU or not in MIG mode
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pHeterogeneousMode* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGpuInstanceGetVgpuSchedulerLog**(
nvmlGpuInstance_t gpuInstance, *nvmlVgpuSchedulerLogInfo_t* *pSchedulerLogInfo,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlGpuInstanceGetVgpuSchedulerLog_v2* instead

Returns the vGPU scheduler logs for the given GPU instance. *pSchedulerLogInfo* points to a caller-allocated structure to contain the logs. The number of elements returned will never exceed *NVML_SCHEDULER_SW_MAX_LOG_ENTRIES*.

To get the entire logs, call the function atleast 5 times a second.

For Blackwell &tm GB20x; or newer fully supported devices.

Parameters

- **gpuInstance** – The GPU instance handle
- **pSchedulerLogInfo** – Reference in which *pSchedulerLogInfo* is written

Returns

- *NVML_SUCCESS* vGPU scheduler logs are successfully obtained
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pSchedulerLogInfo* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pSchedulerLogInfo* is invalid
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlGpuInstanceGetVgpuSchedulerLog_v2**(
nvmlGpuInstance_t gpuInstance, *nvmlVgpuSchedulerLogInfo_v2_t* *pSchedulerLogInfo,
)

Returns the vGPU scheduler logs for the given GPU instance.

pSchedulerLogInfo points to a caller-allocated structure to contain the logs. The number of elements returned will never exceed *NVML_SCHEDULER_SW_MAX_LOG_ENTRIES*.

To get the entire logs, call the function atleast 5 times a second.

For Blackwell &tm GB20x; or newer fully supported devices.

Parameters

- **gpuInstance** – The GPU instance handle
- **pSchedulerLogInfo** – Reference in which *pSchedulerLogInfo* is written

Returns

- *NVML_SUCCESS* vGPU scheduler logs are successfully obtained
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pSchedulerLogInfo* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU

- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlGpuInstanceGetVgpuSchedulerState**(
nvmlGpuInstance_t gpuInstance, *nvmlVgpuSchedulerStateInfo_t* *pSchedulerStateInfo,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlGpuInstanceGetVgpuSchedulerState_v2* instead

Returns the vGPU scheduler state for the given GPU instance. The information returned in *nvmlVgpuSchedulerStateInfo_t* is not relevant if the BEST EFFORT policy is set.

For Blackwell &tm GB20x; or newer fully supported devices.

Parameters

- **gpuInstance** – The GPU instance handle
- **pSchedulerStateInfo** – Reference in which *pSchedulerStateInfo* is returned

Returns

- *NVML_SUCCESS* vGPU scheduler state is successfully obtained
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pSchedulerStateInfo* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pSchedulerStateInfo* is invalid
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlGpuInstanceGetVgpuSchedulerState_v2**(
nvmlGpuInstance_t gpuInstance, *nvmlVgpuSchedulerStateInfo_v2_t* *pSchedulerStateInfo,
)

Returns the vGPU scheduler state for the given GPU instance.

The information returned in nvmlVgpuSchedulerStateInfo_v2_t is not relevant if the BEST EFFORT policy is set.

For Blackwell &tm GB20x; or newer fully supported devices.

Parameters

- **gpuInstance** – The GPU instance handle
- **pSchedulerStateInfo** – Reference in which *pSchedulerStateInfo* is returned

Returns

- *NVML_SUCCESS* vGPU scheduler state is successfully obtained
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pSchedulerStateInfo* is NULL or GPU Instance Id is invalid

- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlGpuInstanceGetVgpuTypeCreatablePlacements**(
nvmlGpuInstance_t gpuInstance, *nvmlVgpuCreatablePlacementInfo_t* *pCreatablePlacementInfo,
)

Query the creatable vGPU placement ID of the vGPU type within a GPU instance.

For Blackwell &tm GB20x; or newer fully supported devices.

An array of creatable vGPU placement IDs for the vGPU type ID indicated by *pCreatablePlacementInfo->vgpuTypeId* is returned in the caller-supplied buffer of *pCreatablePlacementInfo->placementIds*. Memory needed for the placementIds array should be allocated based on maximum instances of a vGPU type per GPU instance which can be queried via *nvmlVgpuTypeGetMaxInstancesPerGpuInstance()*. If the provided count by the caller is insufficient, the function will return NVML_ERROR_INSUFFICIENT_SIZE along with the number of required entries in *pCreatablePlacementInfo->count*. The caller should then reallocate a buffer with the size of pCreatablePlacementInfo->count * sizeof(pCreatablePlacementInfo->placementIds) and invoke the function again. The creatable vGPU placement IDs may differ over time, as there may be restrictions on what type of vGPU the vGPU instance is running.

Parameters

- **gpuInstance** – The GPU instance handle
- **pCreatablePlacementInfo** – Pointer to the list of vGPU creatable placement structure *nvmlVgpuCreatablePlacementInfo_t*

Returns

- *NVML_SUCCESS* Successful completion
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pCreatablePlacementInfo* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_INSUFFICIENT_SIZE* If the buffer is small, element count is returned in *pCreatablePlacementInfo->count*
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pCreatablePlacementInfo* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU or vGPU heterogeneous mode is not enabled
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGpuInstanceSetVgpuHeterogeneousMode**(
nvmlGpuInstance_t gpuInstance, const *nvmlVgpuHeterogeneousMode_t* *pHeterogeneousMode,
)

Enable or disable vGPU heterogeneous mode for the GPU instance.

When in heterogeneous mode, a vGPU can concurrently host timesliced vGPUs with differing framebuffer sizes.

API would return an appropriate error code upon unsuccessful activation. For example, the heterogeneous mode set will fail with error *NVML_ERROR_IN_USE* if any vGPU instance is active

within the GPU instance. The caller of this API is expected to shutdown the vGPU VMs and retry setting the *mode*. On successful return, the function updates the vGPU heterogeneous mode with the user provided *pHeterogeneousMode->mode*. *pHeterogeneousMode->version* is the version number of the structure nvmlVgpuHeterogeneousMode_t, the caller should set the correct version number to set the vGPU heterogeneous mode.

Parameters

- **gpuInstance** – The GPU instance handle
- **pHeterogeneousMode** – Pointer to the caller-provided structure of nvmlVgpuHeterogeneousMode_t

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pHeterogeneousMode* is NULL or *pHeterogeneousMode->mode* is invalid or GPU Instance Id is invalid
- *NVML_ERROR_IN_USE* If the *gpuInstance* is in use
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pHeterogeneousMode* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGpuInstanceSetVgpuSchedulerState**(
nvmlGpuInstance_t gpuInstance, *nvmlVgpuSchedulerState_t* *pScheduler,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlGpuInstanceSetVgpuSchedulerState_v2* instead

Set vGPU scheduler state for the given GPU instance

For Blackwell &tm GB20x; or newer fully supported devices.

Scheduler state and params will be allowed to set only when no VM is running within the GPU instance. In *nvmlVgpuSchedulerState_t*, IFF enableARRMode is enabled then provide the avgFactor and frequency as input. If enableARRMode is disabled then provide timeslice as input.

The scheduler state change won't persist across module load/unload and GPU Instance creation/deletion.

Parameters

- **gpuInstance** – The GPU instance handle
- **pScheduler** – Pointer to the caller-provided structure of nvmlVgpuSchedulerState_t

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pScheduler* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_RESET_REQUIRED* If setting the state failed with fatal error, reboot is required
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU or if any vGPU instance exists
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pScheduler* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGpuInstanceSetVgpuSchedulerState_v2**(
nvmlGpuInstance_t gpuInstance, *nvmlVgpuSchedulerState_v2_t* *pSchedulerState,
)

Set vGPU scheduler state for the given GPU instance.

For Blackwell &tm GB20x; or newer fully supported devices.

Scheduler state and params will be allowed to set only when no VM is running within the GPU instance.

The scheduler state change won't persist across module load/unload and GPU Instance creation/deletion.

Parameters

- **gpuInstance** – The GPU instance handle
- **pSchedulerState** – Pointer to the caller-provided structure of *nvmlVgpuSchedulerState_v2_t*

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_INVALID_ARGUMENT* If *gpuInstance* is NULL or invalid, or *pSchedulerState* is NULL or GPU Instance Id is invalid
- *NVML_ERROR_RESET_REQUIRED* If setting the state failed with fatal error, reboot is required
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU or if any vGPU instance exists
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetEccMode**(
nvmlVgpuInstance_t vgpuInstance, *nvmlEnableState_t* *eccMode,
)

Retrieve the current ECC mode of vGPU instance.

Parameters

- **vgpuInstance** – The identifier of the target vGPU instance
- **eccMode** – Reference in which to return the current ECC mode

Returns

- *NVML_SUCCESS* if the vgpuInstance's ECC mode has been successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *mode* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_NOT_SUPPORTED* if the vGPU doesn't support this feature
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetEncoderCapacity**(
nvmlVgpuInstance_t vgpuInstance, unsigned int *encoderCapacity,
)

Retrieve the encoder capacity of a vGPU instance, as a percentage of maximum encoder capacity with valid values in the range 0-100.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **encoderCapacity** – Reference to an unsigned int for the encoder capacity

Returns

- *NVML_SUCCESS* if *encoderCapacity* has been retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *encoderQueryType* is invalid
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetEncoderSessions**(
nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, *nvmlEncoderSessionInfo_t* *sessionInfo,
)

Retrieves information about all active encoder sessions on a vGPU Instance.

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

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

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **sessionCount** – Reference to caller supplied array size, and returns the number of sessions.
- **sessionInfo** – Reference to caller supplied array in which the list of session information us returned.

Returns

- *NVML_SUCCESS* if *sessionInfo* is fetched
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INSUFFICIENT_SIZE* if *sessionCount* is too small, array element count is returned in *sessionCount*
- *NVML_ERROR_INVALID_ARGUMENT* if *sessionCount* is NULL, or *vgpuInstance* is 0.
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetEncoderStats**(
nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, unsigned int *averageFps, unsigned int *averageLatency,
)

Retrieves the current encoder statistics of a vGPU Instance.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **sessionCount** – Reference to an unsigned int for count of active encoder sessions
- **averageFps** – Reference to an unsigned int for trailing average FPS of all active sessions
- **averageLatency** – Reference to an unsigned int for encode latency in microseconds

Returns

- *NVML_SUCCESS* if *sessionCount*, *averageFps* and *averageLatency* is fetched
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *sessionCount* , or *averageFps* or *averageLatency* is NULL or *vgpuInstance* is 0.
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetFBCSessions**(
nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, *nvmlFBCSessionInfo_t* *sessionInfo,
)

Retrieves information about active frame buffer capture sessions on a vGPU Instance.

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

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

For Maxwell or newer fully supported devices.

Note

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

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **sessionCount** – Reference to caller supplied array size, and returns the number of sessions.
- **sessionInfo** – Reference in which to return the session information

Returns

- *NVML_SUCCESS* if *sessionInfo* is fetched
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *sessionCount* is NULL.
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_INSUFFICIENT_SIZE* if *sessionCount* is too small, array element count is returned in *sessionCount*
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetFBCStats**(
nvmlVgpuInstance_t vgpuInstance, *nvmlFBCStats_t* *fbcStats,
)

Retrieves the active frame buffer capture sessions statistics of a vGPU Instance.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **fbcStats** – Reference to *nvmlFBCStats_t* structure containing NvFBC stats

Returns

- *NVML_SUCCESS* if *fbcStats* is fetched
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *fbcStats* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetFbUsage**(
    nvmlVgpuInstance_t vgpuInstance, unsigned long long *fbUsage,
)

Retrieve the framebuffer usage in bytes.

Framebuffer usage is the amont of vGPU framebuffer memory that is currently in use by the VM.

For Kepler or newer fully supported devices.

Parameters

- **vgpuInstance** – The identifier of the target instance
- **fbUsage** – Pointer to framebuffer usage in bytes

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *fbUsage* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetFrameRateLimit**(
    nvmlVgpuInstance_t vgpuInstance, unsigned int *frameRateLimit,
)

Retrieve the frame rate limit set for the vGPU instance.

Returns the value of the frame rate limit set for the vGPU instance

For Kepler or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **frameRateLimit** – Reference to return the frame rate limit

Returns

- *NVML_SUCCESS* if *frameRateLimit* has been set
- *NVML_ERROR_NOT_SUPPORTED* if frame rate limiter is turned off for the vGPU type
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *frameRateLimit* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetGpuInstanceId**(
nvmlVgpuInstance_t vgpuInstance, unsigned int *gpuInstanceId,
)

Retrieve the GPU Instance ID for the given vGPU Instance.

The API will return a valid GPU Instance ID for MIG backed vGPU Instance, else INVALID_GPU_INSTANCE_ID is returned.

For Kepler or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **gpuInstanceId** – GPU Instance ID

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *gpuInstanceId* is NULL.
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetGpuPciId**(
nvmlVgpuInstance_t vgpuInstance, char *vgpuPciId, unsigned int *length,
)

Retrieves the PCI Id of the given vGPU Instance i.e.

the PCI Id of the GPU as seen inside the VM.

The vGPU PCI id is returned as “00000000:00:00.0” if NVIDIA driver is not installed on the vGPU instance.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **vgpuPciId** – Caller-supplied buffer to return vGPU PCI Id string
- **length** – Size of the vgpuPciId buffer

Returns

- *NVML_SUCCESS* if vGPU PCI Id is successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *vgpuPciId* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_DRIVER_NOT_LOADED* if NVIDIA driver is not running on the vGPU instance
- *NVML_ERROR_INSUFFICIENT_SIZE* if *length* is too small, *length* is set to required length
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetLicenseStatus**(
nvmlVgpuInstance_t vgpuInstance, unsigned int *licensed,
)

Deprecated:

Use *nvmlVgpuInstanceGetLicenseInfo_v2*.

Retrieve the current licensing state of the vGPU instance.

If the vGPU is currently licensed, *licensed* is set to 1, otherwise it is set to 0.

For Kepler or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **licensed** – Reference to return the licensing status

Returns

- *NVML_SUCCESS* if *licensed* has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *licensed* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetMdevUUID**(
nvmlVgpuInstance_t vgpuInstance, char *mdevUuid, unsigned int size,
)

Retrieve the MDEV UUID of a vGPU instance.

The MDEV UUID is a globally unique identifier of the mdev device assigned to the VM, and is returned as a 5-part hexadecimal string, not exceeding 80 characters in length (including the NULL terminator). MDEV UUID is displayed only on KVM platform. See nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **mdevUuid** – Pointer to caller-supplied buffer to hold MDEV UUID
- **size** – Size of buffer in bytes

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_NOT_SUPPORTED* on any hypervisor other than KVM
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *mdevUuid* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_INSUFFICIENT_SIZE* if *size* is too small
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetType**(
nvmlVgpuInstance_t vgpuInstance, *nvmlVgpuTypeId_t* *vgpuTypeId,
)

Retrieve the vGPU type of a vGPU instance.

Returns the vGPU type ID of vgpu assigned to the vGPU instance.

For Kepler or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **vgpuTypeId** – Reference to return the vgpuTypeId

Returns

- *NVML_SUCCESS* if *vgpuTypeId* has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *vgpuTypeId* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetUUID**(
nvmlVgpuInstance_t vgpuInstance, char *uuid, unsigned int size,
)

Retrieve the UUID of a vGPU instance.

The UUID is a globally unique identifier associated with the vGPU, and is returned as a 5-part hexadecimal string, not exceeding 80 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE.

For Kepler or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **uuid** – Pointer to caller-supplied buffer to hold vGPU UUID
- **size** – Size of buffer in bytes

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *uuid* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_INSUFFICIENT_SIZE* if *size* is too small
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetVmDriverVersion**(
    nvmlVgpuInstance_t vgpuInstance, char *version, unsigned int length,
)

Retrieve the NVIDIA driver version installed in the VM associated with a vGPU.

The version is returned as an alphanumeric string in the caller-supplied buffer *version*. The length of the version string will not exceed 80 characters in length (including the NUL terminator). See nvmlConstants::NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE.

nvmlVgpuInstanceGetVmDriverVersion() may be called at any time for a vGPU instance. The guest VM driver version is returned as "Not Available" if no NVIDIA driver is installed in the VM, or the VM has not yet booted to the point where the NVIDIA driver is loaded and initialized.

For Kepler or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **version** – Caller-supplied buffer to return driver version string
- **length** – Size of *version* buffer

Returns

- *NVML_SUCCESS* if *version* has been set
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_INSUFFICIENT_SIZE* if *length* is too small
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetVmID**(
nvmlVgpuInstance_t vgpuInstance, char *vmId, unsigned int size, *nvmlVgpuVmIdType_t* *vmIdType,
)

Retrieve the VM ID associated with a vGPU instance.

The VM ID is returned as a string, not exceeding 80 characters in length (including the NUL terminator). See nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE.

The format of the VM ID varies by platform, and is indicated by the type identifier returned in *vmIdType*.

For Kepler or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **vmId** – Pointer to caller-supplied buffer to hold VM ID
- **size** – Size of buffer in bytes
- **vmIdType** – Pointer to hold VM ID type

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vmId* or *vmIdType* is NULL, or *vgpuInstance* is 0
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_INSUFFICIENT_SIZE* if *size* is too small
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceSetEncoderCapacity**(
nvmlVgpuInstance_t vgpuInstance, unsigned int encoderCapacity,
)

Set the encoder capacity of a vGPU instance, as a percentage of maximum encoder capacity with valid values in the range 0-100.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **encoderCapacity** – Unsigned int for the encoder capacity value

Returns

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

- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *encoderCapacity* is out of range of 0-100.
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetBAR1Info**(
    nvmlVgpuTypeId_t vgpuTypeId, *nvmlVgpuTypeBar1Info_t* *bar1Info,
)

Retrieve the BAR1 info for given vGPU type.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **bar1Info** – Pointer to the vGPU type BAR1 information structure *nvmlVgpuTypeBar1Info_t*

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *bar1Info* is NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetCapabilities**(
    nvmlVgpuTypeId_t vgpuTypeId, *nvmlVgpuCapability_t* capability, unsigned int *capResult,
)

Retrieve the requested capability for a given vGPU type.

Refer to the *nvmlVgpuCapability_t* structure for the specific capabilities that can be queried. The return value in *capResult* should be treated as a boolean, with a non-zero value indicating that the capability is supported.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **capability** – Specifies the *nvmlVgpuCapability_t* to be queried
- **capResult** – A boolean for the queried capability indicating that feature is supported

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *capability* is invalid, or *capResult* is NULL

- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetClass**(
    nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeClass, unsigned int *size,
)

Retrieve the class of a vGPU type.

It will not exceed 64 characters in length (including the NUL terminator). See nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **vgpuTypeClass** – Pointer to string array to return class in
- **size** – Size of string

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *vgpuTypeClass* is NULL
- *NVML_ERROR_INSUFFICIENT_SIZE* if *size* is too small
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetDeviceID**(
    nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *deviceID, unsigned long long *subsystemID,
)

Retrieve the device ID of a vGPU type.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **deviceID** – Device ID and vendor ID of the device contained in single 32 bit value
- **subsystemID** – Subsystem ID and subsystem vendor ID of the device contained in single 32 bit value

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *deviceId* or *subsystemID* are NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetFrameRateLimit**(
    nvmlVgpuTypeId_t vgpuTypeId, unsigned int *frameRateLimit,
)

Retrieve the static frame rate limit value of the vGPU type.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **frameRateLimit** – Reference to return the frame rate limit value

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_NOT_SUPPORTED* if frame rate limiter is turned off for the vGPU type
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *frameRateLimit* is NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetFramebufferSize**(
nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *fbSize,
)

Retrieve the vGPU framebuffer size in bytes.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **fbSize** – Pointer to framebuffer size in bytes

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *fbSize* is NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetGpuInstanceProfileId**(
nvmlVgpuTypeId_t vgpuTypeId, unsigned int *gpuInstanceProfileId,
)

Retrieve the GPU Instance Profile ID for the given vGPU type ID.

The API will return a valid GPU Instance Profile ID for the MIG capable vGPU types, else INVALID_GPU_INSTANCE_PROFILE_ID is returned.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type

- **gpuInstanceProfileId** – GPU Instance Profile ID

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_NOT_SUPPORTED* if *device* is not in vGPU Host virtualization mode
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *gpuInstanceProfileId* is NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetLicense**(
nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeLicenseString, unsigned int size,
)

Retrieve license requirements for a vGPU type.

The license type and version required to run the specified vGPU type is returned as an alphanumeric string, in the form "<license name>,<version>", for example "GRID-Virtual-PC,2.0". If a vGPU is runnable with* more than one type of license, the licenses are delimited by a semicolon, for example "GRID-Virtual-PC,2.0;GRID-Virtual-WS,2.0;GRID-Virtual-WS-Ext,2.0".

The total length of the returned string will not exceed 128 characters, including the NUL terminator. See nvmlVgpuConstants::NVML_GRID_LICENSE_BUFFER_SIZE.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **vgpuTypeLicenseString** – Pointer to buffer to return license info
- **size** – Size of *vgpuTypeLicenseString* buffer

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *vgpuTypeLicenseString* is NULL
- *NVML_ERROR_INSUFFICIENT_SIZE* if *size* is too small
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetMaxInstances**(
nvmlDevice_t device, *nvmlVgpuTypeId_t* vgpuTypeId, unsigned int *vgpuInstanceCount,
)

Retrieve the maximum number of vGPU instances creatable on a device for given vGPU type.

For Kepler or newer fully supported devices.

Parameters

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

- **vgpuTypeId** – Handle to vGPU type
- **vgpuInstanceCount** – Pointer to get the max number of vGPU instances that can be created on a deicve for given vgpuTypeId

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid or is not supported on target device, or *vgpuInstanceCount* is NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetMaxInstancesPerGpuInstance**(
    nvmlVgpuTypeMaxInstance_t *pMaxInstance,
)

Retrieve the maximum number of vGPU instances per GPU instance for given vGPU type.

Parameters

pMaxInstance – Pointer to the caller-provided structure of nvmlVgpuTypeMaxInstance_t

Returns

- *NVML_SUCCESS* Upon success
- *NVML_ERROR_UNINITIALIZED* If library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *pMaxInstance* is NULL or *pMaxInstance->vgpuTypeId* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If not on a vGPU host or an unsupported GPU or non-MIG vGPU type
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *pMaxInstance* is invalid
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetMaxInstancesPerVm**(
    nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCountPerVm,
)

Retrieve the maximum number of vGPU instances supported per VM for given vGPU type.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **vgpuInstanceCountPerVm** – Pointer to get the max number of vGPU instances supported per VM for given *vgpuTypeId*

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized

- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *vgpuInstanceCountPerVm* is NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetName**(
nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeName, unsigned int *size,
)

Retrieve the vGPU type name.

The name is an alphanumeric string that denotes a particular vGPU, e.g. GRID M60-2Q. It will not exceed 64 characters in length (including the NUL terminator). See nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **vgpuTypeName** – Pointer to buffer to return name
- **size** – Size of buffer

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *name* is NULL
- *NVML_ERROR_INSUFFICIENT_SIZE* if *size* is too small
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetNumDisplayHeads**(
nvmlVgpuTypeId_t vgpuTypeId, unsigned int *numDisplayHeads,
)

Retrieve count of vGPU's supported display heads.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **numDisplayHeads** – Pointer to number of display heads

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *numDisplayHeads* is NULL
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuTypeGetResolution**(
nvmlVgpuTypeId_t vgpuTypeId, unsigned int displayIndex, unsigned int *xdim, unsigned int *ydim,
)

Retrieve vGPU display head's maximum supported resolution.

For Kepler or newer fully supported devices.

Parameters

- **vgpuTypeId** – Handle to vGPU type
- **displayIndex** – Zero-based index of display head
- **xdim** – Pointer to maximum number of pixels in X dimension
- **ydim** – Pointer to maximum number of pixels in Y dimension

Returns

- *NVML_SUCCESS* successful completion
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuTypeId* is invalid, or *xdim* or *ydim* are NULL, or *displayIndex* is out of range.
- *NVML_ERROR_UNKNOWN* on any unexpected error

5.2.1.35 vGPU Migration

This chapter describes operations that are associated with vGPU Migration.

5.2.1.35.1 Enumerations

nvmlVgpuPgpuCompatibilityLimitCode_t
vGPU-pGPU compatibility limit codes

nvmlVgpuVmCompatibility_t
vGPU VM compatibility codes

5.2.1.35.2 Functions

nvmlReturn_t *nvmlDeviceGetPgpuMetadataString*(nvmlDevice_t device, char *pgpuMetadata, unsigned int *bufferSize)
Returns the properties of the physical GPU indicated by the device in an ascii-encoded string format.

nvmlReturn_t *nvmlDeviceGetVgpuMetadata*(nvmlDevice_t device, nvmlVgpuPgpuMetadata_t *pgpuMetadata, unsigned int *bufferSize)
Returns a vGPU metadata structure for the physical GPU indicated by *device* .

nvmlReturn_t *nvmlDeviceGetVgpuSchedulerCapabilities*(nvmlDevice_t device, nvmlVgpuSchedulerCapabilities_t *pCapabilities)
Returns the vGPU scheduler capabilities.

nvmlReturn_t *nvmlDeviceGetVgpuSchedulerLog*(nvmlDevice_t device, nvmlVgpuSchedulerLog_t *pSchedulerLog)

nvmlReturn_t *nvmlDeviceGetVgpuSchedulerState*(nvmlDevice_t device, nvmlVgpuSchedulerGetState_t *pSchedulerState)

nvmlReturn_t *nvmlDeviceSetVgpuSchedulerState*(nvmlDevice_t device, nvmlVgpuSchedulerSetState_t *pSchedulerState)

nvmlReturn_t *nvmlGetVgpuCompatibility*(nvmlVgpuMetadata_t *vgpuMetadata, nvmlVgpuPgpuMetadata_t *pgpuMetadata, nvmlVgpuPgpuCompatibility_t *compatibilityInfo)
Takes a vGPU instance metadata structure read from *nvmlVgpuInstanceGetMetadata()* , and a vGPU metadata structure for a physical GPU read from *nvmlDeviceGetVgpuMetadata()* , and returns compatibility information of the vGPU instance and the physical GPU.

nvmlReturn_t *nvmlGetVgpuVersion*(nvmlVgpuVersion_t *supported, nvmlVgpuVersion_t *current)
Query the ranges of supported vGPU versions.

nvmlReturn_t *nvmlSetVgpuVersion*(nvmlVgpuVersion_t *vgpuVersion)
Override the preset range of vGPU versions supported by the NVIDIA vGPU Manager with a range set by an administrator.

nvmlReturn_t *nvmlVgpuInstanceGetMetadata*(nvmlVgpuInstance_t vgpuInstance, nvmlVgpuMetadata_t *vgpuMetadata, unsigned int *bufferSize)
Returns vGPU metadata structure for a running vGPU.

5.2.1.35.3 Structs

nvmlVgpuMetadata_t
vGPU metadata structure.

nvmlVgpuPgpuCompatibility_t
vGPU-pGPU compatibility structure

nvmlVgpuPgpuMetadata_t
Physical GPU metadata structure.

nvmlVgpuVersion_t
Structure representing range of vGPU versions.

5.2.1.35.4 Enumerations

enum **nvmlVgpuPgpuCompatibilityLimitCode_t**
vGPU-pGPU compatibility limit codes

Values:

enumerator **NVML_VGPU_COMPATIBILITY_LIMIT_NONE**
Compatibility is not limited.

enumerator **NVML_VGPU_COMPATIBILITY_LIMIT_HOST_DRIVER**
ompatibility is limited by host driver version.

enumerator **NVML_VGPU_COMPATIBILITY_LIMIT_GUEST_DRIVER**
Compatibility is limited by guest driver version.

enumerator **NVML_VGPU_COMPATIBILITY_LIMIT_GPU**
Compatibility is limited by GPU hardware.

enumerator **NVML_VGPU_COMPATIBILITY_LIMIT_OTHER**
Compatibility is limited by an undefined factor.

enum **nvmlVgpuVmCompatibility_t**
vGPU VM compatibility codes

Values:

enumerator **NVML_VGPU_VM_COMPATIBILITY_NONE**
vGPU is not runnable

enumerator **NVML_VGPU_VM_COMPATIBILITY_COLD**
vGPU is runnable from a cold / powered-off state (ACPI S5)

enumerator **NVML_VGPU_VM_COMPATIBILITY_HIBERNATE**
vGPU is runnable from a hibernated state (ACPI S4)

enumerator **NVML_VGPU_VM_COMPATIBILITY_SLEEP**
vGPU is runnable from a sleeped state (ACPI S3)

enumerator **NVML_VGPU_VM_COMPATIBILITY_LIVE**
vGPU is runnable from a live/paused (ACPI S0)

5.2.1.35.5 Functions

nvmlReturn_t **nvmlDeviceGetPgpuMetadataString**(
nvmlDevice_t device, char *pgpuMetadata, unsigned int *bufferSize,
)

Returns the properties of the physical GPU indicated by the device in an ascii-encoded string format.

The caller passes in a buffer via *pgpuMetadata*, with the size of the buffer in *bufferSize*. If the string is too large to fit in the supplied buffer, the function returns NVML_ERROR_INSUFFICIENT_SIZE with the size needed in *bufferSize*.

Parameters

- **device** – The identifier of the target device
- **pgpuMetadata** – Pointer to caller-supplied buffer into which *pgpuMetadata* is written
- **bufferSize** – Pointer to size of *pgpuMetadata* buffer

Returns

- *NVML_SUCCESS* GPU metadata structure was successfully returned
- *NVML_ERROR_INSUFFICIENT_SIZE* *pgpuMetadata* buffer is too small, required size is returned in *bufferSize*
- *NVML_ERROR_INVALID_ARGUMENT* If *bufferSize* is NULL or *device* is invalid; if *pgpuMetadata* is NULL and the value of *bufferSize* is not 0.

- *NVML_ERROR_NOT_SUPPORTED* If vGPU is not supported by the system
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuMetadata**(
nvmlDevice_t device, *nvmlVgpuPgpuMetadata_t* *pgpuMetadata, unsigned int *bufferSize,
)

Returns a vGPU metadata structure for the physical GPU indicated by *device*.

The structure contains information about the GPU and the currently installed NVIDIA host driver version that's controlling it, together with an opaque data section containing internal state.

The caller passes in a buffer via *pgpuMetadata*, with the size of the buffer in *bufferSize*. If the *pgpuMetadata* structure is too large to fit in the supplied buffer, the function returns NVML_ERROR_INSUFFICIENT_SIZE with the size needed in *bufferSize*.

Parameters

- **device** – The identifier of the target device
- **pgpuMetadata** – Pointer to caller-supplied buffer into which *pgpuMetadata* is written
- **bufferSize** – Pointer to size of *pgpuMetadata* buffer

Returns

- *NVML_SUCCESS* GPU metadata structure was successfully returned
- *NVML_ERROR_INSUFFICIENT_SIZE* pgpuMetadata buffer is too small, required size is returned in *bufferSize*
- *NVML_ERROR_INVALID_ARGUMENT* if *bufferSize* is NULL or *device* is invalid; if *pgpuMetadata* is NULL and the value of *bufferSize* is not 0.
- *NVML_ERROR_NOT_SUPPORTED* vGPU is not supported by the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuSchedulerCapabilities**(
nvmlDevice_t device, *nvmlVgpuSchedulerCapabilities_t* *pCapabilities,
)

Returns the vGPU scheduler capabilities.

The list of supported vGPU schedulers returned in nvmlVgpuSchedulerCapabilities_t is from the NVML_VGPU_SCHEDULER_POLICY_*. This list enumerates the supported scheduler policies if the engine is Graphics type. The other values in nvmlVgpuSchedulerCapabilities_t are also applicable if the engine is Graphics type. For other engine types, it is BEST EFFORT policy. If ARR is supported and enabled, scheduling frequency and averaging factor are applicable else timeSlice is applicable.

For Pascal or newer fully supported devices.

Parameters

- **device** – The identifier of the target *device*
- **pCapabilities** – Reference in which *pCapabilities* is written

Returns

- *NVML_SUCCESS* vGPU scheduler capabilities were successfully obtained
- *NVML_ERROR_INVALID_ARGUMENT* If *pCapabilities* is NULL or *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* The API is not supported in current state or *device* not in vGPU host mode
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuSchedulerLog**(
nvmlDevice_t device, *nvmlVgpuSchedulerLog_t* *pSchedulerLog,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlDeviceGetVgpuSchedulerLog_v2* instead

Returns the vGPU Software scheduler logs. *pSchedulerLog* points to a caller-allocated structure to contain the logs. The number of elements returned will never exceed *NVML_SCHEDULER_SW_MAX_LOG_ENTRIES*.

To get the entire logs, call the function atleast 5 times a second.

For Pascal or newer fully supported devices.

Parameters

- **device** – The identifier of the target *device*
- **pSchedulerLog** – Reference in which *pSchedulerLog* is written

Returns

- *NVML_SUCCESS* vGPU scheduler logs were successfully obtained
- *NVML_ERROR_INVALID_ARGUMENT* If *pSchedulerLog* is NULL or *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* not in vGPU host mode
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuSchedulerState**(
nvmlDevice_t device, *nvmlVgpuSchedulerGetState_t* *pSchedulerState,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlDeviceGetVgpuSchedulerState_v2* instead

Returns the vGPU scheduler state. The information returned in nvmlVgpuSchedulerGetState_t is not relevant if the BEST EFFORT policy is set.

For Pascal or newer fully supported devices.

Parameters

- **device** – The identifier of the target *device*
- **pSchedulerState** – Reference in which *pSchedulerState* is returned

Returns

- *NVML_SUCCESS* vGPU scheduler state is successfully obtained
- *NVML_ERROR_INVALID_ARGUMENT* If *pSchedulerState* is NULL or *device* is invalid
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* not in vGPU host mode
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceSetVgpuSchedulerState**(
nvmlDevice_t device, *nvmlVgpuSchedulerSetState_t* *pSchedulerState,
)

Deprecated:

Will be deprecated in a future release. Use *nvmlDeviceSetVgpuSchedulerState_v2* instead

Sets the vGPU scheduler state.

For Pascal or newer fully supported devices.

The scheduler state change won't persist across module load/unload. Scheduler state and params will be allowed to set only when no VM is running. In nvmlVgpuSchedulerSetState_t, IFF enableARRMode is enabled then provide avgFactorForARR and frequency as input. If enableARRMode is disabled then provide timeslice as input.

Parameters

- **device** – The identifier of the target *device*
- **pSchedulerState** – vGPU *pSchedulerState* to set

Returns

- *NVML_SUCCESS* vGPU scheduler state has been successfully set
- *NVML_ERROR_INVALID_ARGUMENT* If *pSchedulerState* is NULL or *device* is invalid
- *NVML_ERROR_RESET_REQUIRED* If setting *pSchedulerState* failed with fatal error, reboot is required to overcome from this error.
- *NVML_ERROR_NOT_SUPPORTED* If MIG is enabled or *device* not in vGPU host mode or if any vGPU instance currently exists on the *device*
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGetVgpuCompatibility**(
nvmlVgpuMetadata_t *vgpuMetadata, *nvmlVgpuPgpuMetadata_t* *pgpuMetadata, *nvmlVgpuPgpuCompatibility_t* *compatibilityInfo,
)

Takes a vGPU instance metadata structure read from *nvmlVgpuInstanceGetMetadata()*, and a vGPU metadata structure for a physical GPU read from *nvmlDeviceGetVgpuMetadata()*, and returns compatibility information of the vGPU instance and the physical GPU.

The caller passes in a buffer via *compatibilityInfo*, into which a compatibility information structure is written. The structure defines the states in which the vGPU / VM may be booted on the physical GPU. If the vGPU / VM compatibility with the physical GPU is limited, a limit code indicates the factor limiting compatability. (see *nvmlVgpuPgpuCompatibilityLimitCode_t* for details).

Note: vGPU compatibility does not take into account dynamic capacity conditions that may limit a system's ability to boot a given vGPU or associated VM.

Parameters

- **vgpuMetadata** – Pointer to caller-supplied vGPU metadata structure
- **pgpuMetadata** – Pointer to caller-supplied GPU metadata structure
- **compatibilityInfo** – Pointer to caller-supplied buffer to hold compatibility info

Returns

- *NVML_SUCCESS* vGPU metadata structure was successfully returned
- *NVML_ERROR_INVALID_ARGUMENT* If *vgpuMetadata* or *pgpuMetadata* or *bufferSize* are NULL
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlGetVgpuVersion**(
nvmlVgpuVersion_t *supported, *nvmlVgpuVersion_t* *current,
)

Query the ranges of supported vGPU versions.

This function gets the linear range of supported vGPU versions that is preset for the NVIDIA vGPU Manager and the range set by an administrator. If the preset range has not been overridden by *nvmlSetVgpuVersion*, both ranges are the same.

The caller passes pointers to the following *nvmlVgpuVersion_t* structures, into which the NVIDIA vGPU Manager writes the ranges:

a. *supported* structure that represents the preset range of vGPU versions supported by the NVIDIA vGPU Manager.

b. *current* structure that represents the range of supported vGPU versions set by an administrator. By default, this range is the same as the preset range.

Parameters

- **supported** – Pointer to the structure in which the preset range of vGPU versions supported by the NVIDIA vGPU Manager is written
- **current** – Pointer to the structure in which the range of supported vGPU versions set by an administrator is written

Returns

- *NVML_SUCCESS* The vGPU version range structures were successfully obtained.
- *NVML_ERROR_NOT_SUPPORTED* The API is not supported.
- *NVML_ERROR_INVALID_ARGUMENT* The *supported* parameter or the *current* parameter is NULL.
- *NVML_ERROR_UNKNOWN* An error occurred while the data was being fetched.

nvmlReturn_t **nvmlSetVgpuVersion**(*nvmlVgpuVersion_t* *vgpuVersion)

Override the preset range of vGPU versions supported by the NVIDIA vGPU Manager with a range set by an administrator.

This function configures the NVIDIA vGPU Manager with a range of supported vGPU versions set by an administrator. This range must be a subset of the preset range that the NVIDIA vGPU Manager supports. The custom range set by an administrator takes precedence over the preset range and is advertised to the guest VM for negotiating the vGPU version. See *nvmlGetVgpuVersion* for details of how to query the preset range of versions supported.

This function takes a pointer to vGPU version range structure *nvmlVgpuVersion_t* as input to override the preset vGPU version range that the NVIDIA vGPU Manager supports.

After host system reboot or driver reload, the range of supported versions reverts to the range that is preset for the NVIDIA vGPU Manager.

Note

1. The range set by the administrator must be a subset of the preset range that the NVIDIA vGPU Manager supports. Otherwise, an error is returned.
   a. If the range of supported guest driver versions does not overlap the range set by the administrator, the guest driver fails to load.
   b. If the range of supported guest driver versions overlaps the range set by the administrator, the guest driver will load with a negotiated vGPU version that is the maximum value in the overlapping range.
   c. No VMs must be running on the host when this function is called. If a VM is running on the host, the call to this function fails.

Parameters

vgpuVersion – Pointer to a caller-supplied range of supported vGPU versions.

Returns

- *NVML_SUCCESS* The preset range of supported vGPU versions was successfully overridden.
- *NVML_ERROR_NOT_SUPPORTED* The API is not supported.
- *NVML_ERROR_IN_USE* The range was not overridden because a VM is running on the host.
- *NVML_ERROR_INVALID_ARGUMENT* The *vgpuVersion* parameter specifies a range that is outside the range supported by the NVIDIA vGPU Manager or if *vgpuVersion* is NULL.

nvmlReturn_t **nvmlVgpuInstanceGetMetadata**(
nvmlVgpuInstance_t vgpuInstance, *nvmlVgpuMetadata_t* *vgpuMetadata, unsigned int *bufferSize,
)

Returns vGPU metadata structure for a running vGPU.

The structure contains information about the vGPU and its associated VM such as the currently installed NVIDIA guest driver version, together with host driver version and an opaque data section containing internal state.

nvmlVgpuInstanceGetMetadata() may be called at any time for a vGPU instance. Some fields in the returned structure are dependent on information obtained from the guest VM, which may not yet have reached a state where that information is available. The current state of these dependent fields is reflected in the info structure's *nvmlVgpuGuestInfoState_t* field.

The VMM may choose to read and save the vGPU's VM info as persistent metadata associated with the VM, and provide it to Virtual GPU Manager when creating a vGPU for subsequent instances of the VM.

The caller passes in a buffer via *vgpuMetadata*, with the size of the buffer in *bufferSize*. If the vGPU Metadata structure is too large to fit in the supplied buffer, the function returns NVML_ERROR_INSUFFICIENT_SIZE with the size needed in *bufferSize*.

Parameters

- **vgpuInstance** – vGPU instance handle
- **vgpuMetadata** – Pointer to caller-supplied buffer into which vGPU metadata is written
- **bufferSize** – Size of vgpuMetadata buffer

Returns

- *NVML_SUCCESS* vGPU metadata structure was successfully returned
- *NVML_ERROR_INSUFFICIENT_SIZE* vgpuMetadata buffer is too small, required size is returned in *bufferSize*
- *NVML_ERROR_INVALID_ARGUMENT* if *bufferSize* is NULL or *vgpuInstance* is 0; if *vgpuMetadata* is NULL and the value of *bufferSize* is not 0.
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_UNKNOWN* on any unexpected error

5.2.1.36 vGPU Utilization and Accounting

This chapter describes operations that are associated with vGPU Utilization and Accounting.

5.2.1.36.1 Functions

nvmlReturn_t *nvmlDeviceGetVgpuInstancesUtilizationInfo*(nvmlDevice_t device, nvmlVgpuInstancesUtilizationInfo_t *vgpuUtilInfo)
Retrieves recent utilization for vGPU instances running on a physical GPU (device).

nvmlReturn_t *nvmlDeviceGetVgpuProcessUtilization*(nvmlDevice_t device, unsigned long long lastSeenTimeStamp, unsigned int *vgpuProcessSamplesCount, nvmlVgpuProcessUtilizationSample_t *utilizationSamples)
Retrieves current utilization for processes running on vGPUs on a physical GPU (device).

nvmlReturn_t *nvmlDeviceGetVgpuProcessesUtilizationInfo*(nvmlDevice_t device, nvmlVgpuProcessesUtilizationInfo_t *vgpuProcUtilInfo)
Retrieves recent utilization for processes running on vGPU instances on a physical GPU (device).

nvmlReturn_t *nvmlDeviceGetVgpuUtilization*(nvmlDevice_t device, unsigned long long lastSeenTimeStamp, nvmlValueType_t *sampleValType, unsigned int *vgpuInstanceSamplesCount, nvmlVgpuInstanceUtilizationSample_t *utilizationSamples)
Retrieves current utilization for vGPUs on a physical GPU (device).

nvmlReturn_t *nvmlVgpuInstanceClearAccountingPids*(nvmlVgpuInstance_t vgpuInstance)
Clears accounting information of the vGPU instance that have already terminated.

nvmlReturn_t *nvmlVgpuInstanceGetAccountingMode*(nvmlVgpuInstance_t vgpuInstance, nvmlEnableState_t *mode)
Queries the state of per process accounting mode on vGPU.

nvmlReturn_t *nvmlVgpuInstanceGetAccountingPids*(nvmlVgpuInstance_t vgpuInstance, unsigned int *count, unsigned int *pids)
Queries list of processes running on vGPU that can be queried for accounting stats.

nvmlReturn_t *nvmlVgpuInstanceGetAccountingStats*(nvmlVgpuInstance_t vgpuInstance, unsigned int pid, nvmlAccountingStats_t *stats)
Queries process's accounting stats.

nvmlReturn_t *nvmlVgpuInstanceGetLicenseInfo_v2*(nvmlVgpuInstance_t vgpuInstance, nvmlVgpuLicenseInfo_t *licenseInfo)
Query the license information of the vGPU instance.

5.2.1.36.2 Functions

nvmlReturn_t **nvmlDeviceGetVgpuInstancesUtilizationInfo**(
nvmlDevice_t device, *nvmlVgpuInstancesUtilizationInfo_t* *vgpuUtilInfo,
)

Retrieves recent utilization for vGPU instances running on a physical GPU (device).

For Kepler or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, video decoder, jpeg decoder, and OFA for vGPU instances running on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by *vgpuUtilInfo->vgpuUtilArray*. One utilization sample structure is returned per vGPU instance, and includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values in *nvmlValue_t* unions. The function sets the caller-supplied *vgpuUtilInfo->sampleValType* to NVML_VALUE_TYPE_UNSIGNED_INT to indicate the returned value type.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with *vgpuUtilInfo->vgpuUtilArray* set to NULL. The function will return NVML_ERROR_INSUFFICIENT_SIZE, with the current vGPU instance count in *vgpuUtilInfo->vgpuInstanceCount*, or NVML_SUCCESS if the current vGPU instance count is zero. The caller should allocate a buffer of size vgpuUtilInfo->vgpuInstanceCount * sizeof(nvmlVgpuInstanceUtilizationInfo_t). Invoke the function again with the allocated buffer passed in *vgpuUtilInfo->vgpuUtilArray*, and *vgpuUtilInfo->vgpuInstanceCount* set to the number of entries the buffer is sized for.

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

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

Parameters

- **device** – The identifier for the target device

- **vgpuUtilInfo** – Pointer to the caller-provided structure of nvmlVgpuInstanceUtilizationInfo_t

Returns

- *NVML_SUCCESS* If utilization samples are successfully retrieved
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid, *vgpuUtilInfo* is NULL, or *vgpuUtilInfo->vgpuInstanceCount* is 0
- *NVML_ERROR_NOT_SUPPORTED* If vGPU is not supported by the device
- *NVML_ERROR_GPU_IS_LOST* If the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *vgpuUtilInfo* is invalid
- *NVML_ERROR_INSUFFICIENT_SIZE* If *vgpuUtilInfo->vgpuUtilArray* is NULL, or the buffer size of vgpuUtilInfo->vgpuInstanceCount is too small. The caller should check the current vGPU instance count from the returned vgpuUtilInfo->vgpuInstanceCount, and call the function again with a buffer of size vgpuUtilInfo->vgpuInstanceCount * sizeof(nvmlVgpuInstanceUtilizationInfo_t)
- *NVML_ERROR_NOT_FOUND* If sample entries are not found
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuProcessUtilization**(
nvmlDevice_t device, unsigned long long lastSeenTimeStamp, unsigned int *vgpuProcessSamplesCount, *nvmlVgpuProcessUtilizationSample_t* *utilizationSamples,
)

Retrieves current utilization for processes running on vGPUs on a physical GPU (device).

For Maxwell or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder for processes running on vGPU instances active on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by *utilizationSamples*. One utilization sample structure is returned per process running on vGPU instances, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with *utilizationSamples* set to NULL. The function will return NVML_ERROR_INSUFFICIENT_SIZE, with the current vGPU instance count in *vgpuProcessSamplesCount*. The caller should allocate a buffer of size vgpuProcessSamplesCount * sizeof(nvmlVgpuProcessUtilizationSample_t). Invoke the function again with the allocated buffer passed in *utilizationSamples*, and *vgpuProcessSamplesCount* set to the number of entries the buffer is sized for.

On successful return, the function updates *vgpuSubProcessSampleCount* with the number of vGPU sub process utilization sample structures that were actually written. This may differ from a previously read value depending on the number of processes that are active in any given sample period.

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

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

Parameters

- **device** – The identifier for the target device
- **lastSeenTimeStamp** – Return only samples with timestamp greater than lastSeenTimeStamp.
- **vgpuProcessSamplesCount** – Pointer to caller-supplied array size, and returns number of processes running on vGPU instances
- **utilizationSamples** – Pointer to caller-supplied buffer in which vGPU sub process utilization samples are returned

Returns

- *NVML_SUCCESS* if utilization samples are successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, *vgpuProcessSamplesCount* or a sample count of 0 is passed with a non-NULL *utilizationSamples*
- *NVML_ERROR_INSUFFICIENT_SIZE* if supplied *vgpuProcessSamplesCount* is too small to return samples for all vGPU instances currently executing on the device
- *NVML_ERROR_NOT_SUPPORTED* if vGPU is not supported by the device
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_NOT_FOUND* if sample entries are not found
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuProcessesUtilizationInfo**(
    nvmlDevice_t device, *nvmlVgpuProcessesUtilizationInfo_t* *vgpuProcUtilInfo,
)

Retrieves recent utilization for processes running on vGPU instances on a physical GPU (device).

For Maxwell or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, video decoder, jpeg decoder, and OFA for processes running on vGPU instances active on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by *vgpuProcUtilInfo->vgpuProcUtilArray*. One utilization sample structure is returned per process running on vGPU instances, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as “unsigned int” values.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with *vgpuProcUtilInfo->vgpuProcUtilArray* set to NULL. The function will return NVML_ERROR_INSUFFICIENT_SIZE, with the current processes’ count running on vGPU instances in *vgpuProcUtilInfo->vgpuProcessCount*. The caller should allocate a buffer of size vgpuProcUtilInfo->vgpuProcessCount * sizeof(nvmlVgpuProcessUtilizationSample_t). Invoke the function again with the allocated buffer passed in *vgpuProcUtilInfo->vgpuProcUtilArray*, and *vgpuProcUtilInfo->vgpuProcessCount* set to the number of entries the buffer is sized for.

On successful return, the function updates *vgpuProcUtilInfo->vgpuProcessCount* with the number of vGPU sub process utilization sample structures that were actually written. This may differ from a previously read value depending on the number of processes that are active in any given sample period.

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

Parameters

- **device** – The identifier for the target device
- **vgpuProcUtilInfo** – Pointer to the caller-provided structure of nvmlVgpuProcessesUtilizationInfo_t

Returns

- *NVML_SUCCESS* If utilization samples are successfully retrieved
- *NVML_ERROR_UNINITIALIZED* If the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* If *device* is invalid, or *vgpuProcUtilInfo* is null
- *NVML_ERROR_ARGUMENT_VERSION_MISMATCH* If the version of *vgpuProcUtilInfo* is invalid
- *NVML_ERROR_INSUFFICIENT_SIZE* If *vgpuProcUtilInfo->vgpuProcUtilArray* is null, or supplied *vgpuProcUtilInfo->vgpuProcessCount* is too small to return samples for all processes on vGPU instances currently executing on the device. The caller should check the current processes count from the returned *vgpuProcUtilInfo->vgpuProcessCount*, and call the function again with a buffer of size vgpuProcUtilInfo->vgpuProcessCount * sizeof(nvmlVgpuProcessUtilizationSample_t)
- *NVML_ERROR_NOT_SUPPORTED* If vGPU is not supported by the device
- *NVML_ERROR_GPU_IS_LOST* If the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_NOT_FOUND* If sample entries are not found
- *NVML_ERROR_UNKNOWN* On any unexpected error

nvmlReturn_t **nvmlDeviceGetVgpuUtilization**(
nvmlDevice_t device, unsigned long long lastSeenTimeStamp, *nvmlValueType_t* *sampleValType, unsigned int *vgpuInstanceSamplesCount, *nvmlVgpuInstanceUtilizationSample_t* *utilizationSamples,
)

Retrieves current utilization for vGPUs on a physical GPU (device).

For Kepler or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder for vGPU instances running on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by *utilizationSamples*. One utilization sample structure is returned per vGPU instance, and includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "un-

signed int" values in *nvmlValue_t* unions. The function sets the caller-supplied *sampleValType* to NVML_VALUE_TYPE_UNSIGNED_INT to indicate the returned value type.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with *utilizationSamples* set to NULL. The function will return NVML_ERROR_INSUFFICIENT_SIZE, with the current vGPU instance count in *vgpuInstanceSamplesCount*, or NVML_SUCCESS if the current vGPU instance count is zero. The caller should allocate a buffer of size vgpuInstanceSamplesCount * sizeof(nvmlVgpuInstanceUtilizationSample_t). Invoke the function again with the allocated buffer passed in *utilizationSamples*, and *vgpuInstanceSamplesCount* set to the number of entries the buffer is sized for.

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

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

Parameters

- **`device`** – The identifier for the target device
- **`lastSeenTimeStamp`** – Return only samples with timestamp greater than lastSeenTimeStamp.
- **`sampleValType`** – Pointer to caller-supplied buffer to hold the type of returned sample values
- **`vgpuInstanceSamplesCount`** – Pointer to caller-supplied array size, and returns number of vGPU instances
- **`utilizationSamples`** – Pointer to caller-supplied buffer in which vGPU utilization samples are returned

Returns

- *NVML_SUCCESS* if utilization samples are successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *device* is invalid, *vgpuInstanceSamplesCount* or *sampleValType* is NULL, or a sample count of 0 is passed with a non-NULL *utilizationSamples*
- *NVML_ERROR_INSUFFICIENT_SIZE* if supplied *vgpuInstanceSamplesCount* is too small to return samples for all vGPU instances currently executing on the device
- *NVML_ERROR_NOT_SUPPORTED* if vGPU is not supported by the device
- *NVML_ERROR_GPU_IS_LOST* if the target GPU has fallen off the bus or is otherwise inaccessible
- *NVML_ERROR_NOT_FOUND* if sample entries are not found
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **`nvmlVgpuInstanceClearAccountingPids`**(
    nvmlVgpuInstance_t vgpuInstance,
)

Clears accounting information of the vGPU instance that have already terminated.

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

Note

Accounting Mode needs to be on. See *nvmlVgpuInstanceGetAccountingMode*.

Note

Only compute and graphics applications stats are reported and can be cleared since monitoring applications stats don't contribute to GPU utilization.

Parameters

vgpuInstance – The identifier of the target vGPU instance

Returns

- *NVML_SUCCESS* if accounting information has been cleared
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is invalid
- *NVML_ERROR_NO_PERMISSION* if the user doesn't have permission to perform this operation
- *NVML_ERROR_NOT_SUPPORTED* if the vGPU doesn't support this feature or accounting mode is disabled
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetAccountingMode**(
    nvmlVgpuInstance_t vgpuInstance, *nvmlEnableState_t* *mode,
)

Queries the state of per process accounting mode on vGPU.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – The identifier of the target vGPU instance
- **mode** – Reference in which to return the current accounting mode

Returns

- *NVML_SUCCESS* if the mode has been successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *mode* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system

- *NVML_ERROR_NOT_SUPPORTED* if the vGPU doesn't support this feature
- *NVML_ERROR_DRIVER_NOT_LOADED* if NVIDIA driver is not running on the vGPU instance
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetAccountingPids**(
    nvmlVgpuInstance_t vgpuInstance, unsigned int *count, unsigned int *pids,
)

Queries list of processes running on vGPU that can be queried for accounting stats.

The list of processes returned can be in running or terminated state.

For Maxwell or newer fully supported devices.

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

For more details see *nvmlVgpuInstanceGetAccountingStats*.

See also

nvmlVgpuInstanceGetAccountingPids

Note

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

Parameters

- **vgpuInstance** – The identifier of the target vGPU instance
- **count** – Reference in which to provide the *pids* array size, and to return the number of elements ready to be queried
- **pids** – Reference in which to return list of process ids

Returns

- *NVML_SUCCESS* if pids were successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *count* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_NOT_SUPPORTED* if the vGPU doesn't support this feature or accounting mode is disabled
- *NVML_ERROR_INSUFFICIENT_SIZE* if *count* is too small (*count* is set to expected value)

- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetAccountingStats**(
nvmlVgpuInstance_t vgpuInstance, unsigned int pid, *nvmlAccountingStats_t* *stats,
)

Queries process's accounting stats.

For Maxwell or newer fully supported devices.

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

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

Note

Accounting Mode needs to be on. See *nvmlVgpuInstanceGetAccountingMode*.

Note

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

Note

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

Parameters

- **vgpuInstance** – The identifier of the target vGPU instance
- **pid** – Process Id of the target process to query stats for
- **stats** – Reference in which to return the process's accounting stats

Returns

- *NVML_SUCCESS* if stats have been successfully retrieved
- *NVML_ERROR_UNINITIALIZED* if the library has not been successfully initialized
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *stats* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system or *stats* is not found
- *NVML_ERROR_NOT_SUPPORTED* if the vGPU doesn't support this feature or accounting mode is disabled
- *NVML_ERROR_UNKNOWN* on any unexpected error

nvmlReturn_t **nvmlVgpuInstanceGetLicenseInfo_v2**(
nvmlVgpuInstance_t vgpuInstance, *nvmlVgpuLicenseInfo_t* *licenseInfo,
)

Query the license information of the vGPU instance.

For Maxwell or newer fully supported devices.

Parameters

- **vgpuInstance** – Identifier of the target vGPU instance
- **licenseInfo** – Pointer to vGPU license information structure

Returns

- *NVML_SUCCESS* if information is successfully retrieved
- *NVML_ERROR_INVALID_ARGUMENT* if *vgpuInstance* is 0, or *licenseInfo* is NULL
- *NVML_ERROR_NOT_FOUND* if *vgpuInstance* does not match a valid active vGPU instance on the system
- *NVML_ERROR_DRIVER_NOT_LOADED* if NVIDIA driver is not running on the vGPU instance
- *NVML_ERROR_UNKNOWN* on any unexpected error

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

CPER (Common Platform Error Record)
Types and API for retrieving CPER data.

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

Confidential Computing definitions

Constants

Device Commands
This chapter describes NVML operations that change the state of the device.

Device Enums

Device Queries
This chapter describes that queries that NVML can perform against each device.

Device Structs

Drain State definitions

Drain states
This chapter describes methods that NVML can perform against each device to control their drain state and recognition by NVML and NVIDIA kernel driver.

Encoder Structs

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

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

Event Types
Event Types which user can be notified about.

Excluded GPU Queries
This chapter describes NVML operations that are associated with excluded GPUs.

Fabric definitions

Field Value Enums

Field Value Queries
This chapter describes NVML operations that are associated with retrieving Field Values from NVML.

Frame Buffer Capture Structures

GPM Enums

GPM Functions

GPM Structs

Initialization and Cleanup
NVML Methods that handle the NVML Library initialization and cleanup.

MPS Locality Domain Device Enumeration
On supported Linux systems, set the following environment variable before the first NVML initialization to enumerate the device view reported by MPS:

Multi Instance GPU Management
This chapter describes NVML operations that are associated with Multi Instance GPU management.

NVML GPM

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

NvmlClocksEventReasons

PRM Access
This chapter describes NVML operations that are associated with PRM register reads.

Power Profile Information

Power Smoothing Information

System Queries
This chapter describes the queries that NVML can perform against the local system.

Unit Commands
This chapter describes NVML operations that change the state of the unit.

Unit Queries
This chapter describes that queries that NVML can perform against each unit.

Unit Structs

vGPU APIs
This chapter describes operations that are associated with NVIDIA vGPU Software products.

vGPU Constants

vGPU Enums

vGPU Enums, Constants, Structs

vGPU Management
This chapter describes APIs supporting NVIDIA vGPU.

vGPU Migration
This chapter describes operations that are associated with vGPU Migration.

vGPU Structs

vGPU Utilization and Accounting
This chapter describes operations that are associated with vGPU Utilization and Accounting.

5.2.2. Pages

5.2.2.1 Deprecated List

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 *nvmlClocksThrottleReasonAll*
Use *nvmlClocksEventReasonAll* instead

Global *nvmlClocksThrottleReasonApplicationsClocksSetting*

Global *nvmlClocksThrottleReasonDisplayClockSetting*
Use *nvmlClocksEventReasonDisplayClockSetting* instead

Global *nvmlClocksThrottleReasonGpuIdle*
Use *nvmlClocksEventReasonGpuIdle* instead

Global *nvmlClocksThrottleReasonNone*

Use *nvmlClocksEventReasonNone* instead

Global *nvmlClocksThrottleReasonSwPowerCap*

Use *nvmlClocksEventReasonSwPowerCap* instead

Global *nvmlClocksThrottleReasonSwThermalSlowdown*

Use *nvmlClocksEventReasonSwThermalSlowdown* instead

Global *nvmlClocksThrottleReasonSyncBoost*

Use *nvmlClocksEventReasonSyncBoost* instead

Global *nvmlClocksThrottleReasonUserDefinedClocks*

Renamed to *nvmlClocksThrottleReasonApplicationsClocksSetting* as the name describes the situation more accurately.

Global *nvmlDeviceFreezeNvLinkUtilizationCounter* (nvmlDevice_t device, unsigned int link, unsigned int counter, nvmlEnableState_t freeze)

Freezing NVLINK utilization counters is no longer supported.

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

Applications clocks are deprecated and will be removed in CUDA 14.0.

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

Use *nvmlDeviceGetCurrentClocksEventReasons* instead

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

Applications clocks are deprecated and will be removed in CUDA 14.0.

Global *nvmlDeviceGetDetailedEccErrors* (nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, *nvmlEccErrorCounts_t* *eccCounts)

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

Global *nvmlDeviceGetGpuFabricInfo* (nvmlDevice_t device, *nvmlGpuFabricInfo_t* *gpuFabricInfo)

Will be deprecated in a future release. Use *nvmlDeviceGetGpuFabricInfoV* instead

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

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 *nvmlDeviceGetNvLinkUtilizationControl* (nvmlDevice_t device, unsigned int link, unsigned int counter, *nvmlNvLinkUtilizationControl_t* *control)

Getting utilization counter control is no longer supported.

Global *nvmlDeviceGetNvLinkUtilizationCounter* (nvmlDevice_t device, unsigned int link, unsigned int counter, unsigned long long *rxcounter, unsigned long long *txcounter)

Use *nvmlDeviceGetFieldValues* with NVML_FI_DEV_NVLINK_THROUGHPUT_* as field values instead.

Global *nvmlDeviceGetPowerManagementMode* (nvmlDevice_t device, nvmlEnableState_t *mode)

This API has been deprecated.

Global *nvmlDeviceGetPowerState* (nvmlDevice_t device, nvmlPstates_t *pState)

Use *nvmlDeviceGetPerformanceState*. This function exposes an incorrect generalization.

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

Use *nvmlDeviceGetSupportedClocksEventReasons* instead

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

Use *nvmlDeviceGetTemperatureV* instead

Global *nvmlDeviceGetVgpuSchedulerLog* (nvmlDevice_t device, *nvmlVgpuSchedulerLog_t* *pSchedulerLog)

Will be deprecated in a future release. Use *nvmlDeviceGetVgpuSchedulerLog_v2* instead

Global *nvmlDeviceGetVgpuSchedulerState* (nvmlDevice_t device, *nvmlVgpuSchedulerGetState_t* *pSchedulerState)

Will be deprecated in a future release. Use *nvmlDeviceGetVgpuSchedulerState_v2* instead

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

Use *nvmlDeviceGetFieldValues* to query this data. This API will be removed in CUDA 14.0.

Global *nvmlDeviceResetApplicationsClocks* (nvmlDevice_t device)

Applications clocks are deprecated and will be removed in CUDA 14.0.

Global *nvmlDeviceResetNvLinkUtilizationCounter* (nvmlDevice_t device, unsigned int link, unsigned int counter)

Resetting NVLINK utilization counters is no longer supported.

Global *nvmlDeviceSetApplicationsClocks* (nvmlDevice_t device, unsigned int memClockMHz, unsigned int graphicsClockMHz)

Applications clocks are deprecated and will be removed in CUDA 14.0.

Global *nvmlDeviceSetGpcClkVfOffset* (nvmlDevice_t device, int offset)

Will be deprecated in a future release. Use *nvmlDeviceSetClockOffsets* instead. It works on Maxwell onwards GPU architectures.

Global *nvmlDeviceSetMemClkVfOffset* (nvmlDevice_t device, int offset)

Will be deprecated in a future release. Use *nvmlDeviceSetClockOffsets* instead. It works on Maxwell onwards GPU architectures.

Global *nvmlDeviceSetNvLinkUtilizationControl* (nvmlDevice_t device, unsigned int link, unsigned int counter, *nvmlNvLinkUtilizationControl_t* *control, unsigned int reset)

Setting utilization counter control is no longer supported.

Global *nvmlDeviceSetVgpuSchedulerState* (nvmlDevice_t device, *nvmlVgpuSchedulerSetState_t* *pSchedulerState)

Will be deprecated in a future release. Use *nvmlDeviceSetVgpuSchedulerState_v2* instead

Global *nvmlDeviceWorkloadPowerProfileClearRequestedProfiles* (nvmlDevice_t device, nvmlWorkloadPowerProfileRequestedProfiles_t *requestedProfiles)

Use *nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1* instead Clear Requested Performance Profiles

Global *nvmlDeviceWorkloadPowerProfileSetRequestedProfiles* (nvmlDevice_t device, nvmlWorkloadPowerProfileRequestedProfiles_t *requestedProfiles)

Use *nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1* instead Set Requested Performance Profiles

Global *nvmlEccBitType_t*

See *nvmlMemoryErrorType_t* for a more flexible type

Class *nvmlEccErrorCounts_t*

Different GPU families can have different memory error counters See *nvmlDeviceGetMemoryErrorCounter*

Class *nvmlGpuFabricInfo_v2_t*

nvmlGpuFabricInfo_v2_t is deprecated and will be removed in a future release. Use *nvmlGpuFabricInfo_v3_t* instead

Class *nvmlGpuFabricInfo_v3_t*

nvmlGpuFabricInfo_v3_t is deprecated and will be removed in a future release. Use *nvmlGpuFabricInfo_v4_t* instead.

Global *nvmlGpuInstanceGetVgpuSchedulerLog* (nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerLogInfo_t *pSchedulerLogInfo)

Will be deprecated in a future release. Use *nvmlGpuInstanceGetVgpuSchedulerLog_v2* instead

Global *nvmlGpuInstanceGetVgpuSchedulerState* (nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerStateInfo_t *pSchedulerStateInfo)

Will be deprecated in a future release. Use *nvmlGpuInstanceGetVgpuSchedulerState_v2* instead

Global *nvmlGpuInstanceSetVgpuSchedulerState* (nvmlGpuInstance_t gpuInstance, nvmlVgpuSchedulerState_t *pScheduler)

Will be deprecated in a future release. Use *nvmlGpuInstanceSetVgpuSchedulerState_v2* instead

Class *nvmlPlatformInfo_v1_t*

The *nvmlPlatformInfo_v1_t* will be deprecated in the subsequent releases. Use *nvmlPlatformInfo_v2_t*

Global *nvmlVgpuInstanceGetLicenseStatus* (nvmlVgpuInstance_t vgpuInstance, unsigned int *licensed)

Use *nvmlVgpuInstanceGetLicenseInfo_v2*.

Deprecated List

5.2.3. Structs

5.2.3.1 nvmlAccountingStats_t

struct **nvmlAccountingStats_t**

Describes accounting statistics of a process.

Public Members

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.

5.2.3.2 nvmlAccountingStats_v2_t

struct **nvmlAccountingStats_v2_t**

Describes accounting statistics (v2) of a process.

Public Members

unsigned int **pid**

Process Id of the target process to query stats for.

unsigned int **isRunning**

Flag to represent if the process is running (1 for running, 0 for terminated)

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 int **sampleCount**

The sample counts since the process starts.

unsigned long long **sumGpuUtil**

The sum of process's GR engine utilization in unit of pct * 100.

unsigned long long **sumFbUtil**

The sum of process's FB bandwidth utilization in unit of pct * 100.

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.

5.2.3.3 nvmlActiveVgpuInstanceInfo_v1_t

struct **nvmlActiveVgpuInstanceInfo_v1_t**

Structure to store active vGPU instance information — Version 1.

Public Members

unsigned int **version**

IN: The version number of this struct.

unsigned int **vgpuCount**

IN/OUT: Count of the active vGPU instances.

nvmlVgpuInstance_t ***vgpuInstances**

IN/OUT: list of active vGPU instances.

5.2.3.4 nvmlAdaptiveTgpModeInfo_v1_t

struct **nvmlAdaptiveTgpModeInfo_v1_t**

Stores Adaptive TGP Mode state and telemetry for a GPU.

Public Members

nvmlEnableState_t **inBandEnableRequest**

[out] In-band enable requested (NVML_FEATURE_ENABLED) or not requested (NVML_FEATURE_DISABLED)

nvmlEnableState_t **featureAllowedByAdmin**

[out] Feature allowed by out-of-band/admin (NVML_FEATURE_ENABLED) or not allowed (NVML_FEATURE_DISABLED)

nvmlEnableState_t **adminOverrideEnabled**

[out] Out-of-band/admin override active (NVML_FEATURE_ENABLED) or inactive (NVML_FEATURE_DISABLED)

nvmlEnableState_t **enablementStatus**

[out] Enablement after arbitration: active (NVML_FEATURE_ENABLED) or inactive (NVML_FEATURE_DISABLED)

unsigned int **adjustedLimitMw**

[out] Adjusted TGP limit in milliwatts (valid only when feature is enabled)

5.2.3.5 nvmlBAR1Memory_t

struct **nvmlBAR1Memory_t**

BAR1 Memory allocation Information for a device.

Public Members

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)

5.2.3.6 nvmlBBXTimeData_v1_t

struct **nvmlBBXTimeData_v1_t**

BBX Time Data.

Public Members

unsigned int **timeRun**

[out] Cumulative number of seconds the GPU has had the driver loaded

5.2.3.7 nvmlBridgeChipHierarchy_t

struct **nvmlBridgeChipHierarchy_t**

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.

Public Members

unsigned char **bridgeCount**

Number of Bridge Chips on the Board.

nvmlBridgeChipInfo_t **bridgeChipInfo**[(128)]

Hierarchy of Bridge Chips on the board.

5.2.3.8 nvmlBridgeChipInfo_t

struct **nvmlBridgeChipInfo_t**

Information about the Bridge Chip Firmware.

Public Members

nvmlBridgeChipType_t **type**

Type of Bridge Chip.

unsigned int **fwVersion**

Firmware Version. 0=Version is unavailable.

5.2.3.9 nvmlC2cModeInfo_v1_t

struct **nvmlC2cModeInfo_v1_t**

C2C Mode information for a device.

Public Members

unsigned int **isC2cEnabled**

5.2.3.10 nvmlCPERCursor_v1_t

struct **nvmlCPERCursor_v1_t**

CPER query and cursor parameters.

Groups the filter options and opaque read position used to iterate through CPER records.

Cursor semantics for iterative calls:

- Pass the same nvmlCPERCursor_v1_t (i.e. the same *cursor*) on every call in a single iteration sequence.
- Do not modify this struct between calls.
- To change *cperTypeMask* or *uuid*, set *handle* to *NVML_CPER_CURSOR_HANDLE_INIT* and call again with the updated filter (this starts a new iteration).

Public Members

unsigned int **cperTypeMask**

[IN] Types of records to access. Bitmask of *nvmlCPERType_t* values. To change, reset *handle* to *NVML_CPER_CURSOR_HANDLE_INIT*.

char **uuid**[80]

[IN] UUID of target to filter records for. Required for *NVML_CPER_ACCESS_TYPE_GPU*. To change, reset *handle* to *NVML_CPER_CURSOR_HANDLE_INIT*.

nvmlCPERCursorHandle_t **handle**

[IN/OUT] Opaque handle tracking read position. Initialize to *NVML_CPER_CURSOR_HANDLE_INIT* on first call; pass the same nvmlCPERCursor_v1_t on the next call to continue. Caller must not interpret or modify.

5.2.3.11 nvmlClkMonFaultInfo_t

struct **nvmlClkMonFaultInfo_t**

Clock Monitor error types.

Public Members

unsigned int **clkApiDomain**

The clock domain that faulted, encoded as a legacy 1-hot bitmask with one bit set.

The bit is set as per the clock domain identifier. For example, a value of 0x4 (bit index 2) corresponds to clock domain identifier 2.

unsigned int **clkDomainFaultMask**

Faults Information.

5.2.3.12 nvmlClkMonStatus_t

struct **nvmlClkMonStatus_t**

Clock Monitor Status.

Public Members

unsigned int **bGlobalStatus**

Fault status Indicator.

unsigned int **clkMonListSize**
Total faulted domain numbers.

nvmlClkMonFaultInfo_t **clkMonList**[32]
The fault Information structure.

5.2.3.13 nvmlClockOffset_v1_t

struct **nvmlClockOffset_v1_t**
Clock offset info.

Public Members

unsigned int **version**
The version number of this struct.

nvmlClockType_t **type**

nvmlPstates_t **pstate**

int **clockOffsetMHz**

int **minClockOffsetMHz**

int **maxClockOffsetMHz**

5.2.3.14 nvmlComputeInstanceInfo_t

struct **nvmlComputeInstanceInfo_t**

Public Members

nvmlDevice_t **device**
Parent device.

nvmlGpuInstance_t **gpuInstance**
Parent GPU instance.

unsigned int **id**
Unique instance ID within the GPU instance.

unsigned int **profileId**
Unique profile ID within the GPU instance.

nvmlComputeInstancePlacement_t **placement**
Placement for this instance within the GPU instance's compute slice range {0, sliceCount}.

5.2.3.15 nvmlComputeInstancePlacement_t

struct **nvmlComputeInstancePlacement_t**

Public Members

unsigned int **start**
Index of first occupied compute slice.

unsigned int **size**
Number of compute slices occupied.

5.2.3.16 nvmlComputeInstanceProfileInfo_t

struct **nvmlComputeInstanceProfileInfo_t**
Compute instance profile information.

Public Members

unsigned int **id**
Unique profile ID within the GPU instance.

unsigned int **sliceCount**
GPU Slice count.

unsigned int **instanceCount**
Compute instance count.

unsigned int **multiprocessorCount**
Streaming Multiprocessor count.

unsigned int **sharedCopyEngineCount**
Shared Copy Engine count.

unsigned int **sharedDecoderCount**
Shared Decoder Engine count.

unsigned int **sharedEncoderCount**
Shared Encoder Engine count.

unsigned int **sharedJpegCount**
Shared JPEG Engine count.

unsigned int **sharedOfaCount**

Shared OFA Engine count.

5.2.3.17 nvmlComputeInstanceProfileInfo_v2_t

struct **nvmlComputeInstanceProfileInfo_v2_t**

Compute instance profile information (v2).

Version 2 adds the *nvmlComputeInstanceProfileInfo_v2_t::version* field to the start of the structure, and the *nvmlComputeInstanceProfileInfo_v2_t::name* field to the end. This structure is not backwards-compatible with *nvmlComputeInstanceProfileInfo_t*.

Public Members

unsigned int **version**

Structure version identifier (set to *nvmlComputeInstanceProfileInfo_v2*)

unsigned int **id**

Unique profile ID within the GPU instance.

unsigned int **sliceCount**

GPU Slice count.

unsigned int **instanceCount**

Compute instance count.

unsigned int **multiprocessorCount**

Streaming Multiprocessor count.

unsigned int **sharedCopyEngineCount**

Shared Copy Engine count.

unsigned int **sharedDecoderCount**

Shared Decoder Engine count.

unsigned int **sharedEncoderCount**

Shared Encoder Engine count.

unsigned int **sharedJpegCount**

Shared JPEG Engine count.

unsigned int **sharedOfaCount**

Shared OFA Engine count.

char **name**[96]

Profile name.

5.2.3.18 nvmlComputeInstanceProfileInfo_v3_t

struct **nvmlComputeInstanceProfileInfo_v3_t**

Compute instance profile information (v3).

Version 3 adds the *nvmlComputeInstanceProfileInfo_v3_t::capabilities* field *nvmlComputeInstanceProfileInfo_t*.

Public Members

unsigned int **version**

Structure version identifier (set to *nvmlComputeInstanceProfileInfo_v3*)

unsigned int **id**

Unique profile ID within the GPU instance.

unsigned int **sliceCount**

GPU Slice count.

unsigned int **instanceCount**

Compute instance count.

unsigned int **multiprocessorCount**

Streaming Multiprocessor count.

unsigned int **sharedCopyEngineCount**

Shared Copy Engine count.

unsigned int **sharedDecoderCount**

Shared Decoder Engine count.

unsigned int **sharedEncoderCount**

Shared Encoder Engine count.

unsigned int **sharedJpegCount**

Shared JPEG Engine count.

unsigned int **sharedOfaCount**

Shared OFA Engine count.

char **name**[96]

Profile name.

unsigned int **capabilities**

Additional capabilities.

5.2.3.19 nvmlConfComputeGetKeyRotationThresholdInfo_v1_t

struct **nvmlConfComputeGetKeyRotationThresholdInfo_v1_t**

Public Members

unsigned int **version**

unsigned long long **attackerAdvantage**

5.2.3.20 nvmlConfComputeGpuAttestationReport_t

struct **nvmlConfComputeGpuAttestationReport_t**

Public Members

unsigned int **isCecAttestationReportPresent**
output

unsigned int **attestationReportSize**
output

unsigned int **cecAttestationReportSize**
output

unsigned char **nonce**[0x20]
input: spdm supports 32 bytes on nonce

unsigned char **attestationReport**[0x2000]
output

unsigned char **cecAttestationReport**[0x1000]
output

5.2.3.21 nvmlConfComputeGpuCertificate_t

struct **nvmlConfComputeGpuCertificate_t**

Public Members

unsigned int **certChainSize**

unsigned int **attestationCertChainSize**

unsigned char **certChain**[0x1000]

unsigned char **attestationCertChain**[0x1400]

5.2.3.22 nvmlConfComputeMemSizeInfo_t

struct **nvmlConfComputeMemSizeInfo_t**

Protected memory size.

Public Members

unsigned long long **protectedMemSizeKib**

unsigned long long **unprotectedMemSizeKib**

5.2.3.23 nvmlConfComputeSetKeyRotationThresholdInfo_v1_t

struct **nvmlConfComputeSetKeyRotationThresholdInfo_v1_t**

Public Members

unsigned int **version**

unsigned long long **maxAttackerAdvantage**

5.2.3.24 nvmlConfComputeSystemCaps_t

struct **nvmlConfComputeSystemCaps_t**

Public Members

unsigned int **cpuCaps**

unsigned int **gpusCaps**

5.2.3.25 nvmlConfComputeSystemState_t

struct **nvmlConfComputeSystemState_t**

Public Members

unsigned int **environment**

unsigned int **ccFeature**

unsigned int **devToolsMode**

5.2.3.26 nvmlCoolerInfo_v1_t

struct **nvmlCoolerInfo_v1_t**

Public Members

unsigned int **version**
the API version number

unsigned int **index**
the cooler index

nvmlCoolerControl_t **signalType**
OUT: the cooler's control signal characteristics.

nvmlCoolerTarget_t **target**
OUT: the target that cooler cools.

5.2.3.27 nvmlCoreRailMetrics_t

struct **nvmlCoreRailMetrics_t**
Metrics for all core rails in the system.

Public Members

nvmlRailMetrics_t **rails**[2]
Array of core rail metrics.

5.2.3.28 nvmlDeviceAddressingMode_v1_t

struct **nvmlDeviceAddressingMode_v1_t**
Struct to represent device addressing mode information.

Public Members

unsigned int **version**
API version.

unsigned int **value**
One of *nvmlDeviceAddressingModeType_t*.

5.2.3.29 nvmlDeviceAttributes_t

struct **nvmlDeviceAttributes_t**

Public Members

unsigned int **multiprocessorCount**
Streaming Multiprocessor count.

unsigned int **sharedCopyEngineCount**
Shared Copy Engine count.

unsigned int **sharedDecoderCount**
Shared Decoder Engine count.

unsigned int **sharedEncoderCount**
Shared Encoder Engine count.

unsigned int **sharedJpegCount**
Shared JPEG Engine count.

unsigned int **sharedOfaCount**
Shared OFA Engine count.

unsigned int **gpuInstanceSliceCount**
GPU instance slice count.

unsigned int **computeInstanceSliceCount**
Compute instance slice count.

unsigned long long **memorySizeMB**
Device memory size (in MiB)

5.2.3.30 nvmlDeviceCapabilities_v1_t

struct **nvmlDeviceCapabilities_v1_t**
Device capabilities.

Public Members

unsigned int **version**
the API version number

unsigned int **capMask**
OUT: Bit mask of capabilities.

5.2.3.31 nvmlDeviceCurrentClockFreqs_v1_t

struct **nvmlDeviceCurrentClockFreqs_v1_t**
Device current clocks string.

Public Members

unsigned int **version**
the API version number

char **str**[2048]
OUT: the current clock frequency string.

5.2.3.32 nvmlDevicePerfModes_v1_t

struct **nvmlDevicePerfModes_v1_t**
Device performance modes string.

Public Members

unsigned int **version**
the API version number

char **str**[2048]
OUT: the performance modes string.

5.2.3.33 nvmlDevicePowerMizerModes_v1_t

struct **nvmlDevicePowerMizerModes_v1_t**

Public Members

unsigned int **currentMode**
OUT: the current powermizer mode.

unsigned int **mode**
IN: the powermizer mode to set.

unsigned int **supportedPowerMizerModes**
The bitmask of supported power mizer modes on this device.

The supported modes can be combined using the bitwise OR operator '|'. For example, if a device supports all PowerMizer modes, the bitmask would be: supportedPowerMizerModes = ((1 << NVML_POWER_MIZER_MODE_ADAPTIVE) | (1 << NVML_POWER_MIZER_MODE_PREFER_MAXIMUM_PERFORMANCE) | (1 << NVML_POWER_MIZER_MODE_AUTO) | (1 << NVML_POWER_MIZER_MODE_PREFER_CONSISTENT_PERFOR

This bitmask can be used to check which power mizer modes are available on the device by performing a bitwise AND operation with the specific mode you want to check. OUT: Bitmask of supported powermizer modes

5.2.3.34 nvmlDramEncryptionInfo_v1_t

struct **nvmlDramEncryptionInfo_v1_t**

DRAM Encryption Info.

Public Members

unsigned int **version**

IN - the API version number.

nvmlEnableState_t **encryptionState**

IN/OUT - DRAM Encryption state.

5.2.3.35 nvmlEccBankRemapperHistogram_v1_t

struct **nvmlEccBankRemapperHistogram_v1_t**

Structure to store bank remapper histogram.

Public Members

unsigned int **maxSpareGroupCount**

Number of groups that have maximum spare.

unsigned int **noSpareGroupCount**

Number of groups that have not spare.

5.2.3.36 nvmlEccBankRemapperStatus_v1_t

struct **nvmlEccBankRemapperStatus_v1_t**

Structure to store bank remapper status.

Public Members

unsigned int **activeRemappings**

Number of active remappings.

unsigned int **inactiveRemappings**

Number of inactive remappings.

unsigned int **bPending**

Whether there exists any pending bank remapping. 0 for no pending remapping, 1 for pending remapping.

nvmlEccBankRemapperHistogram_v1_t **histogram**

Bank remapper histogram.

5.2.3.37 nvmlEccErrorCounts_t

struct **nvmlEccErrorCounts_t**

Detailed ECC error counts for a device.

Deprecated:

Different GPU families can have different memory error counters See *nvmlDeviceGetMemoryErrorCounter*

Public Members

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.

5.2.3.38 nvmlEccSramErrorStatus_v1_t

struct **nvmlEccSramErrorStatus_v1_t**

Structure to store SRAM uncorrectable error counters.

Public Members

unsigned int **version**

the API version number

unsigned long long **aggregateUncParity**
aggregate uncorrectable parity error count

unsigned long long **aggregateUncSecDed**
aggregate uncorrectable SEC-DED error count

unsigned long long **aggregateCor**
aggregate correctable error count

unsigned long long **volatileUncParity**
volatile uncorrectable parity error count

unsigned long long **volatileUncSecDed**
volatile uncorrectable SEC-DED error count

unsigned long long **volatileCor**
volatile correctable error count

unsigned long long **aggregateUncBucketL2**
aggregate uncorrectable error count for L2 cache bucket

unsigned long long **aggregateUncBucketSm**
aggregate uncorrectable error count for SM bucket

unsigned long long **aggregateUncBucketPcie**
aggregate uncorrectable error count for PCIE bucket

unsigned long long **aggregateUncBucketMcu**
aggregate uncorrectable error count for Microcontroller bucket

unsigned long long **aggregateUncBucketOther**
aggregate uncorrectable error count for Other bucket

unsigned int **bThresholdExceeded**
if the error threshold of field diag is exceeded

5.2.3.39 nvmlEccSramUniqueUncorrectedErrorCounts_v1_t

struct **nvmlEccSramUniqueUncorrectedErrorCounts_v1_t**

Public Members

unsigned int **version**
the API version number

unsigned int **entryCount**
: the number of error count entries

nvmlEccSramUniqueUncorrectedErrorEntry_v1_t ***entries**
: pointer to caller-supplied buffer to return the SRAM unique uncorrected ECC error count entries

5.2.3.40 nvmlEccSramUniqueUncorrectedErrorEntry_v1_t

struct **nvmlEccSramUniqueUncorrectedErrorEntry_v1_t**

Public Members

unsigned int **unit**
: the SRAM unit index

unsigned int **location**
: the error location within the SRAM unit

unsigned int **sublocation**
: the error sublocation within the SRAM unit

unsigned int **extlocation**
: the error extlocation within the SRAM unit

unsigned int **address**
: the error address within the SRAM unit

unsigned int **isParity**
: if the SRAM error is parity or not

unsigned int **count**
: the error count at the same SRAM address

5.2.3.41 nvmlEncoderSessionInfo_t

struct **nvmlEncoderSessionInfo_t**
: Structure to hold encoder session data.

Public Members

unsigned int **sessionId**
: Unique session ID.

unsigned int **pid**
: Owning process ID.

nvmlVgpuInstance_t **vgpuInstance**
: Owning vGPU instance ID (only valid on vGPU hosts, otherwise zero)

nvmlEncoderType_t **codecType**
: Video encoder type.

unsigned int **hResolution**
: Current encode horizontal resolution.

unsigned int **vResolution**
: Current encode vertical resolution.

unsigned int **averageFps**
: Moving average encode frames per second.

unsigned int **averageLatency**
: Moving average encode latency in microseconds.

5.2.3.42 nvmlEventData_t

struct **nvmlEventData_t**
: Information about occurred event.

Public Members

nvmlDevice_t **device**
: Specific device where the event occurred.

unsigned long long **eventType**
: Information about what specific event occurred.

unsigned long long **eventData**
: Stores Xid error for the device in the event of nvmlEventTypeXidCriticalError,.

unsigned int **gpuInstanceId**
: If MIG is enabled and nvmlEventTypeXidCriticalError event is attributable to a GPU.

unsigned int **computeInstanceId**
: If MIG is enabled and nvmlEventTypeXidCriticalError event is attributable to a.

5.2.3.43 nvmlEventSetGetContextCount_v1_t

struct **nvmlEventSetGetContextCount_v1_t**

Parameters for retrieving the number of context records associated with the most recent event.

Public Members

unsigned int **count**

[out] Number of context records associated with the most recent event.

5.2.3.44 nvmlEventSetGetContextData_v1_t

struct **nvmlEventSetGetContextData_v1_t**

Parameters for retrieving raw context data associated with the most recent event.

Public Members

void ***data**

[in] Optional caller-owned buffer that receives the raw context payload.

unsigned int **index**

[in] Zero-based context index.

unsigned int **dataSize**

[in,out] Size of `data` on input; actual or required size on output.

5.2.3.45 nvmlEventSetGetContextInfo_v1_t

struct **nvmlEventSetGetContextInfo_v1_t**

Parameters for retrieving context metadata associated with the most recent event.

Public Members

unsigned int **index**

[in] Zero-based context index.

unsigned int **nvmlGpuOperationalEventContextType**

[out] *nvmlGpuOperationalEventContextType_t* value describing the NVML public interpretation of the context payload.

unsigned int **sourceEventContextType**

[out] Source-defined context payload type identifier carried by the event.

unsigned int **dataSize**

[out] Context payload size in bytes, excluding alignment padding.

unsigned short **dataFormatVersion**

[out] Payload format version for `sourceEventContextType`.

5.2.3.46 nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t

struct **nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t**

Parameters for retrieving decoded GPU legacy-Xid context data.

Public Members

unsigned int **index**

[in] Zero-based context index.

unsigned int **xidCode**

[out] Legacy Xid code carried in a GPU Operational Event context.

5.2.3.47 nvmlEventSetWait_v3_t

struct **nvmlEventSetWait_v3_t**

Parameters for waiting on an event set with *nvmlEventSetWait_v3*.

This structure supplies the wait timeout and returns either NVML event-bit or structured event data. For NVML event-bit data, `dataType` is *NVML_EVENT_DATA_TYPE_NVML_EVENT*, `eventType` contains the NVML event bit, fields such as `eventData`, `gpuInstanceId`, and `computeInstanceId` preserve existing semantics, and structured-only fields are set to 0 or empty values. For structured GPU Operational Events, `dataType` is *NVML_EVENT_DATA_TYPE_GPU_OPERATIONAL_EVENT*, `eventType` is set to *nvmlEventTypeNone*, and the structured metadata fields are populated.

Clients that subscribe to both NVML event-bit and structured formats on the same event set should branch on `dataType` to determine which format was returned. During the transition period, the same underlying incident may generate both an NVML event-bit notification and a structured notification.

Public Members

unsigned int **timeoutMs**

[in] Maximum amount of time to wait, in milliseconds.

unsigned int **dataType**

[out] *nvmlEventDataType_t* value indicating which event-data format is populated.

char **uuid**[96]

[out] UUID for the GPU where the event occurred. Empty if unavailable.

char **sourceModule**[16]

[out] Source module signature for structured events. Not guaranteed to be NULL-terminated. Empty for NVML event-bit events.

unsigned long long **eventType**

[out] NVML event bit for *NVML_EVENT_DATA_TYPE_NVML_EVENT* events; *nvmlEventTypeNone* for structured events.

unsigned long long **eventData**

[out] Xid code for *nvmlEventTypeXidCriticalError*, or 0 when not applicable.

unsigned long long **groupCursor**

[out] Structured event group identifier. 0 for NVML event-bit events.

unsigned long long **instanceId**

[out] Structured event sequence identifier. 0 for NVML event-bit events.

unsigned long long **timestampUsec**

[out] Event timestamp in microseconds. 0 if unavailable.

unsigned long long **traceId**

[out] Structured event trace identifier. 0 for NVML event-bit events.

unsigned int **gpuInstanceId**

[out] MIG GPU instance ID for NVML event-bit data, or `NVML_GPU_INSTANCE_ID_ANY` when not applicable.

unsigned int **computeInstanceId**

[out] MIG compute instance ID for NVML event-bit data, or `NVML_COMPUTE_INSTANCE_ID_ANY` when not applicable.

unsigned int **severity**

[out] *nvmlOperationalEventSeverity_t* value for structured events. May contain newer severity values not named in this header. `NVML_OPERATIONAL_EVENT_SEVERITY_ALL` for NVML event-bit events.

unsigned int **categoryId**

[out] Source-defined structured event category identifier. 0 for NVML event-bit events.

unsigned int **moduleEventCode**

[out] Source-module-defined event code. Interpret with `sourceModule`. 0 for NVML event-bit events.

unsigned int **scope**

[out] Structured event scope identifier. 0 for NVML event-bit events.

unsigned int **originator**

[out] Structured event originator identifier. 0 for NVML event-bit events.

unsigned int **moduleInstance**

[out] Structured event module instance identifier. 0 for NVML event-bit events.

unsigned int **chipletId**

[out] Structured event chiplet identifier. 0 for NVML event-bit events.

unsigned int **logLevel**

[out] *nvmlGpuOperationalEventLogLevel_t* value for structured GPU Operational Events. May contain newer log-level values not named in this header. `NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_ALL` for NVML event-bit events.

unsigned int **attributes**

[out] Bitmask of `NVML_OPERATIONAL_EVENT_ATTR_*` values for structured events. May contain newer bits not named in this header. 0 for NVML event-bit events.

unsigned int **groupCperSize**

[out] Associated CPER record size in bytes. 0 when unavailable.

unsigned int **groupAttributes**

[out] Bitmask of `NVML_OPERATIONAL_EVENT_GROUP_ATTR_*` values for structured events. May contain newer bits not named in this header. 0 for NVML event-bit events.

unsigned char **groupSize**

[out] Total number of events in the structured event group. 0 for NVML event-bit events.

unsigned char **groupIndex**

[out] Zero-based index within the structured event group. 0 for NVML event-bit events.

5.2.3.48 nvmlExcludedDeviceInfo_t

struct **nvmlExcludedDeviceInfo_t**

Excluded GPU device information.

Public Members

nvmlPciInfo_t **pciInfo**

The PCI information for the excluded GPU.

char **uuid**[80]

The ASCII string UUID for the excluded GPU.

5.2.3.49 nvmlFBCSessionInfo_t

struct **nvmlFBCSessionInfo_t**

Structure to hold FBC session data.

Public Members

unsigned int **sessionId**
: Unique session ID.

unsigned int **pid**
: Owning process ID.

nvmlVgpuInstance_t **vgpuInstance**
: Owning vGPU instance ID (only valid on vGPU hosts, otherwise zero)

unsigned int **displayOrdinal**
: Display identifier.

nvmlFBCSessionType_t **sessionType**
: Type of frame buffer capture session.

unsigned int **sessionFlags**
: Session flags (one or more of NVML_NVFBC_SESSION_FLAG_XXX).

unsigned int **hMaxResolution**
: Max horizontal resolution supported by the capture session.

unsigned int **vMaxResolution**
: Max vertical resolution supported by the capture session.

unsigned int **hResolution**
: Horizontal resolution requested by caller in capture call.

unsigned int **vResolution**
: Vertical resolution requested by caller in capture call.

unsigned int **averageFPS**
: Moving average new frames captured per second.

unsigned int **averageLatency**
: Moving average new frame capture latency in microseconds.

5.2.3.50 nvmlFBCStats_t

struct **nvmlFBCStats_t**
: Structure to hold frame buffer capture sessions stats.

Public Members

unsigned int **sessionsCount**

Total no of sessions.

unsigned int **averageFPS**

Moving average new frames captured per second.

unsigned int **averageLatency**

Moving average new frame capture latency in microseconds.

5.2.3.51 nvmlFanSpeedInfo_v1_t

struct **nvmlFanSpeedInfo_v1_t**

Fan speed info.

Public Members

unsigned int **version**

the API version number

unsigned int **fan**

the fan index

unsigned int **speed**

OUT: the fan speed in RPM.

5.2.3.52 nvmlFieldValue_t

struct **nvmlFieldValue_t**

Information for a Field Value Sample.

Public Members

unsigned int **fieldId**

ID of the NVML field to retrieve. This must be set before any call that uses this struct. See the constants starting with NVML_FI_ above.

unsigned int **scopeId**

Scope ID can represent data used by NVML depending on fieldId's context. For example, for NVLink throughput counter data, scopeId can represent linkId.

long long **timestamp**

CPU Timestamp of this value in microseconds since 1970.

long long **latencyUsec**

How long this field value took to update (in usec) within NVML. This may be averaged across several fields that are serviced by the same driver call.

nvmlValueType_t **valueType**

Type of the value stored in value.

nvmlReturn_t **nvmlReturn**

Return code for retrieving this value. This must be checked before looking at value, as value is undefined if nvmlReturn != NVML_SUCCESS.

nvmlValue_t **value**

Value for this field. This is only valid if nvmlReturn == NVML_SUCCESS.

5.2.3.53 nvmlGetCPER_v1_t

struct **nvmlGetCPER_v1_t**

Input/output structure for *nvmlSystemGetCPER_v1*.

Public Members

nvmlCPERCursor_v1_t **cursor**

[IN/OUT] Query parameters and cursor. See *nvmlCPERCursor_v1_t*

unsigned char ***buffer**

[OUT] Buffer to be filled (allocated by client). May be NULL for size query.

unsigned int **bufferSize**

[IN/OUT] Size of *buffer*. Set to 0 with *buffer* NULL to query required size. On return, set to required or used size; 0 means no (more) records.

5.2.3.54 nvmlGetMemoryLimits_v1_t

struct **nvmlGetMemoryLimits_v1_t**

Describes the current memory limits that are set for the device.

This structure holds the necessary information that can be used to get the current soft and hard memory limits of a device.

The softLimit is the amount of memory that is guaranteed before allocations may fail due to memory pressure.

The hardLimit is the maximum amount of memory that can be allocated.

This should be cross-referenced with *nvmlDeviceGetMemoryInfo_v2* while in that cgroup to see how much memory is actually available to allocate for the device.

No limit is set when softLimit is 0 and hardLimit is *NVML_DEVICE_MEMORY_LIMIT_MAX*.

Public Members

const char ***nameSpace**

[in] Full path to sysfs cgroup file name

unsigned long long **softLimit**

[out] Currently set soft memory limit in Bytes.

unsigned long long **hardLimit**

[out] Currently set hard memory limit in Bytes.

unsigned long long **currentUsed**

[out] Currently used memory in Bytes.

5.2.3.55 nvmlGpmMetric_t

struct **nvmlGpmMetric_t**

GPM metric information.

Public Members

unsigned int **metricId**

IN: NVML_GPM_METRIC_? define of which metric to retrieve.

nvmlReturn_t **nvmlReturn**

OUT: Status of this metric. If this is nonzero, then value is not valid.

double **value**

OUT: Value of this metric. Is only valid if nvmlReturn is 0 (NVML_SUCCESS)

char ***shortName**

char ***longName**

char ***unit**

struct *nvmlGpmMetric_t*::[anonymous] **metricInfo**

OUT: Metric name and unit. Those can be NULL if not defined.

5.2.3.56 nvmlGpmMetricsGet_t

struct **nvmlGpmMetricsGet_t**

GPM buffer information.

Public Members

unsigned int **version**
IN: Set to NVML_GPM_METRICS_GET_VERSION.

unsigned int **numMetrics**
IN: How many metrics to retrieve in metrics[].

nvmlGpmSample_t **sample1**
IN: Sample buffer.

nvmlGpmSample_t **sample2**
IN: Sample buffer.

nvmlGpmMetric_t **metrics**[*NVML_GPM_METRIC_MAX*]
IN/OUT: Array of metrics. Set metricId on call. See nvmlReturn and value on return.

5.2.3.57 nvmlGpmSupport_t

struct **nvmlGpmSupport_t**
GPM device information.

Public Members

unsigned int **version**
IN: Set to NVML_GPM_SUPPORT_VERSION.

unsigned int **isSupportedDevice**
OUT: Indicates device support.

5.2.3.58 nvmlGpuDynamicPstatesInfo_t

struct **nvmlGpuDynamicPstatesInfo_t**

Public Members

unsigned int **flags**
Reserved for future use.

unsigned int **bIsPresent**
Set if this utilization domain is present on this GPU.

unsigned int **percentage**
Percentage of time where the domain is considered busy in the last 1-second interval.

unsigned int **incThreshold**

Utilization threshold that can trigger a perf-increasing P-State change when crossed.

unsigned int **decThreshold**

Utilization threshold that can trigger a perf-decreasing P-State change when crossed.

struct *nvmlGpuDynamicPstatesInfo_t*::[anonymous] **utilization**[8]

5.2.3.59 nvmlGpuFabricClique_v1_t

struct **nvmlGpuFabricClique_v1_t**

Fabric clique entry.

Each entry represents a single (type, id) pair describing a clique assignment for a given fabric operation type.

Public Members

unsigned char **type**

Clique type. See NVML_GPU_FABRIC_CLIQUE_TYPE_*.

unsigned int **id**

Clique ID assigned by the Fabric Manager.

5.2.3.60 nvmlGpuFabricInfo_t

struct **nvmlGpuFabricInfo_t**

Contains the device fabric information.

Public Members

unsigned char **clusterUuid**[16]

Uuid of the cluster to which this GPU belongs.

nvmlReturn_t **status**

Error status, if any. Must be checked only if state returns “complete”.

unsigned int **cliqueId**

ID of the fabric clique to which this GPU belongs.

nvmlGpuFabricState_t **state**

Current state of GPU registration process. See NVML_GPU_FABRIC_STATE_*.

5.2.3.61 nvmlGpuFabricInfo_v2_t

struct **nvmlGpuFabricInfo_v2_t**

GPU Fabric information (v2).

Deprecated:

nvmlGpuFabricInfo_v2_t is deprecated and will be removed in a future release. Use *nvmlGpuFabricInfo_v3_t* instead

Version 2 adds the *nvmlGpuFabricInfo_v2_t::version* field to the start of the structure, and the *nvmlGpuFabricInfo_v2_t::healthMask* field to the end. This structure is not backwards-compatible with *nvmlGpuFabricInfo_t*.

Public Members

unsigned int **version**

Structure version identifier (set to nvmlGpuFabricInfo_v2)

unsigned char **clusterUuid**[16]

Uuid of the cluster to which this GPU belongs.

nvmlReturn_t **status**

Probe Error status, if any. Must be checked only if Probe state returns "complete".

unsigned int **cliqueId**

ID of the fabric clique to which this GPU belongs.

nvmlGpuFabricState_t **state**

Current Probe State of GPU registration process. See NVML_GPU_FABRIC_STATE_*.

unsigned int **healthMask**

GPU Fabric health Status Mask. See NVML_GPU_FABRIC_HEALTH_MASK_*.

5.2.3.62 nvmlGpuFabricInfo_v3_t

struct **nvmlGpuFabricInfo_v3_t**

GPU Fabric information (v3).

Deprecated:

nvmlGpuFabricInfo_v3_t is deprecated and will be removed in a future release. Use *nvmlGpuFabricInfo_v4_t* instead.

Public Members

unsigned int **version**

Structure version identifier (set to nvmlGpuFabricInfo_v3)

unsigned char **clusterUuid**[16]

Uuid of the cluster to which this GPU belongs.

nvmlReturn_t **status**

Probe Error status, if any. Must be checked only if Probe state returns "complete".

unsigned int **cliqueId**

ID of the fabric clique to which this GPU belongs.

nvmlGpuFabricState_t **state**

Current Probe State of GPU registration process. See NVML_GPU_FABRIC_STATE_*.

unsigned int **healthMask**

GPU Fabric health Status Mask. See NVML_GPU_FABRIC_HEALTH_MASK_*.

unsigned char **healthSummary**

GPU Fabric health summary. See NVML_GPU_FABRIC_HEALTH_SUMMARY_*.

5.2.3.63 nvmlGpuFabricInfo_v4_t

struct **nvmlGpuFabricInfo_v4_t**

GPU Fabric information (v4).

Extends v3 by replacing the single *cliqueId* field with a flat array of (type, id) clique entries. The legacy v3 *cliqueId* maps to the *id* of the first *NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_POINTER* entry.

Public Members

unsigned char **clusterUuid**[16]

Uuid of the cluster to which this GPU belongs.

nvmlReturn_t **status**

Probe Error status, if any. Must be checked only if state returns "complete".

nvmlGpuFabricClique_v1_t **cliques**[64]

Clique entries, sorted by ascending type then ascending id.

unsigned int **numCliques**

Number of valid entries in *cliques*[].

nvmlGpuFabricState_t **state**

Current Probe State. See NVML_GPU_FABRIC_STATE_*.

unsigned int **healthMask**
GPU Fabric health Status Mask. See NVML_GPU_FABRIC_HEALTH_MASK_*.

unsigned char **healthSummary**
GPU Fabric health summary. See NVML_GPU_FABRIC_HEALTH_SUMMARY_*.

5.2.3.64 nvmlGpuInstanceInfo_t

struct **nvmlGpuInstanceInfo_t**

Public Members

nvmlDevice_t **device**
Parent device.

unsigned int **id**
Unique instance ID within the device.

unsigned int **profileId**
Unique profile ID within the device.

nvmlGpuInstancePlacement_t **placement**
Placement for this instance.

5.2.3.65 nvmlGpuInstancePlacement_t

struct **nvmlGpuInstancePlacement_t**

Public Members

unsigned int **start**
Index of first occupied memory slice.

unsigned int **size**
Number of memory slices occupied.

5.2.3.66 nvmlGpuInstanceProfileInfo_t

struct **nvmlGpuInstanceProfileInfo_t**
GPU instance profile information.

Public Members

unsigned int **id**

Unique profile ID within the device.

unsigned int **isP2pSupported**

Peer-to-Peer support.

unsigned int **sliceCount**

GPU Slice count.

unsigned int **instanceCount**

GPU instance count.

unsigned int **multiprocessorCount**

Streaming Multiprocessor count.

unsigned int **copyEngineCount**

Copy Engine count.

unsigned int **decoderCount**

Decoder Engine count.

unsigned int **encoderCount**

Encoder Engine count.

unsigned int **jpegCount**

JPEG Engine count.

unsigned int **ofaCount**

OFA Engine count.

unsigned long long **memorySizeMB**

Memory size in MBytes.

5.2.3.67 nvmlGpuInstanceProfileInfo_v2_t

struct **nvmlGpuInstanceProfileInfo_v2_t**

GPU instance profile information (v2).

Version 2 adds the *nvmlGpuInstanceProfileInfo_v2_t::version* field to the start of the structure, and the *nvmlGpuInstanceProfileInfo_v2_t::name* field to the end. This structure is not backwards-compatible with *nvmlGpuInstanceProfileInfo_t*.

Public Members

unsigned int **version**
Structure version identifier (set to *nvmlGpuInstanceProfileInfo_v2*)

unsigned int **id**
Unique profile ID within the device.

unsigned int **isP2pSupported**
Peer-to-Peer support.

unsigned int **sliceCount**
GPU Slice count.

unsigned int **instanceCount**
GPU instance count.

unsigned int **multiprocessorCount**
Streaming Multiprocessor count.

unsigned int **copyEngineCount**
Copy Engine count.

unsigned int **decoderCount**
Decoder Engine count.

unsigned int **encoderCount**
Encoder Engine count.

unsigned int **jpegCount**
JPEG Engine count.

unsigned int **ofaCount**
OFA Engine count.

unsigned long long **memorySizeMB**
Memory size in MBytes.

char **name**[96]
Profile name.

5.2.3.68 nvmlGpuInstanceProfileInfo_v3_t

struct **nvmlGpuInstanceProfileInfo_v3_t**
GPU instance profile information (v3).

Version 3 removes isP2pSupported field and adds the *nvmlGpuInstanceProfileInfo_v3_t::capabilities* field *nvmlGpuInstanceProfileInfo_t*.

Public Members

unsigned int **version**
: Structure version identifier (set to *nvmlGpuInstanceProfileInfo_v3*)

unsigned int **id**
: Unique profile ID within the device.

unsigned int **sliceCount**
: GPU Slice count.

unsigned int **instanceCount**
: GPU instance count.

unsigned int **multiprocessorCount**
: Streaming Multiprocessor count.

unsigned int **copyEngineCount**
: Copy Engine count.

unsigned int **decoderCount**
: Decoder Engine count.

unsigned int **encoderCount**
: Encoder Engine count.

unsigned int **jpegCount**
: JPEG Engine count.

unsigned int **ofaCount**
: OFA Engine count.

unsigned long long **memorySizeMB**
: Memory size in MBytes.

char **name**[96]
: Profile name.

unsigned int **capabilities**
: Additional capabilities.

5.2.3.69 nvmlGpuOperationalEventConfig_v1_t

struct **nvmlGpuOperationalEventConfig_v1_t**
: Configuration for registering structured GPU Operational Events with *nvmlEventSetRegisterGpuOperationalEvents_v1*.

Initialize the structure to zero and then set `uuid` to select the target GPU. Default values register new device-wide events with log-level and severity filters disabled.

Public Members

char **uuid**[96]

[in] Target GPU UUID string. Must be a NULL-terminated "GPU-..." UUID.

unsigned int **minLogLevel**

[in] *nvmlGpuOperationalEventLogLevel_t* value for the minimum GPU Operational Event log level. `NVML_GPU_OPERATIONAL_EVENT_LOG_LEVEL_ALL` means no filter.

unsigned int **minSeverity**

[in] *nvmlOperationalEventSeverity_t* value for the minimum Operational Event severity threshold. `NVML_OPERATIONAL_EVENT_SEVERITY_ALL` means no filter.

5.2.3.70 nvmlGpuThermalSettings_t

struct **nvmlGpuThermalSettings_t**

Struct to hold the thermal sensor settings.

Public Members

unsigned int **count**

nvmlThermalController_t **controller**

int **defaultMinTemp**

int **defaultMaxTemp**

int **currentTemp**

nvmlThermalTarget_t **target**

struct *nvmlGpuThermalSettings_t*::[anonymous] **sensor**[3]

5.2.3.71 nvmlGpuUptime_v1_t

struct **nvmlGpuUptime_v1_t**

GPU Uptime.

Public Members

unsigned int **uptime**

[out] Number of seconds the GPU has been up since it was initialized by the driver

5.2.3.72 nvmlGridLicensableFeature_t

struct **nvmlGridLicensableFeature_t**

Structure containing vGPU software licensable feature information.

Public Members

nvmlGridLicenseFeatureCode_t **featureCode**

Licensed feature code.

unsigned int **featureState**

Non-zero if feature is currently licensed, otherwise zero.

char **licenseInfo**[128]

Deprecated.

char **productName**[128]

Product name of feature.

unsigned int **featureEnabled**

Non-zero if feature is enabled, otherwise zero.

nvmlGridLicenseExpiry_t **licenseExpiry**

License expiry structure containing date and time.

5.2.3.73 nvmlGridLicensableFeatures_t

struct **nvmlGridLicensableFeatures_t**

Structure to store vGPU software licensable features.

Public Members

int **isGridLicenseSupported**

Non-zero if vGPU Software Licensing is supported on the system, otherwise zero.

unsigned int **licensableFeaturesCount**

Entries returned in *gridLicensableFeatures* array.

nvmlGridLicensableFeature_t **gridLicensableFeatures**[3]

Array of vGPU software licensable features.

5.2.3.74 nvmlGridLicenseExpiry_t

struct **nvmlGridLicenseExpiry_t**

Structure to store license expiry date and time values.

Public Members

unsigned int **year**

Year value of license expiry.

unsigned short **month**

Month value of license expiry.

unsigned short **day**

Day value of license expiry.

unsigned short **hour**

Hour value of license expiry.

unsigned short **min**

Minutes value of license expiry.

unsigned short **sec**

Seconds value of license expiry.

unsigned char **status**

License expiry status.

5.2.3.75 nvmlHostname_v1_t

struct **nvmlHostname_v1_t**

Stores the hostname string assigned to a GPU device.

Public Members

char **value**[64]

null-terminated hostname string

5.2.3.76 nvmlHwbcEntry_t

struct **nvmlHwbcEntry_t**

Description of HWBC entry.

Public Members

unsigned int **hwbcId**

char **firmwareVersion**[32]

5.2.3.77 nvmlLedState_t

struct **nvmlLedState_t**

LED states for an S-class unit.

Public Members

char **cause**[256]

If amber, a text description of the cause.

nvmlLedColor_t **color**

GREEN or AMBER.

5.2.3.78 nvmlMarginTemperature_v1_t

struct **nvmlMarginTemperature_v1_t**

Margin temperature values.

Public Members

unsigned int **version**

The version number of this struct.

int **marginTemperature**

The margin temperature value.

5.2.3.79 nvmlMask255_t

struct **nvmlMask255_t**

Public Members

unsigned int **mask**[8]

5.2.3.80 nvmlMemory_t

struct **nvmlMemory_t**

Memory allocation information for a device (v1).

The total amount is equal to the sum of the amounts of free and used memory.

Public Members

unsigned long long **total**

Total physical device memory (in bytes)

unsigned long long **free**

Unallocated device memory (in bytes)

unsigned long long **used**

Sum of Reserved and Allocated device memory (in bytes). Note that the driver/GPU always sets aside a small amount of memory for bookkeeping.

5.2.3.81 nvmlMemory_v2_t

struct **nvmlMemory_v2_t**

Memory allocation information for a device (v2).

Version 2 adds versioning for the struct and the amount of system-reserved memory as an output.

Public Members

unsigned int **version**

Structure format version (must be 2)

unsigned long long **total**

Total physical device memory (in bytes)

unsigned long long **reserved**

Device memory (in bytes) reserved for system use (driver or firmware)

unsigned long long **free**

Unallocated device memory (in bytes)

unsigned long long **used**

Allocated device memory (in bytes).

5.2.3.82 nvmlNvLinkInfo_v1_t

struct **nvmlNvLinkInfo_v1_t**

Struct to represent per device NVLINK information v1.

Public Members

unsigned int **version**

IN - the API version number.

unsigned int **isNvleEnabled**
OUT - NVLINK encryption enablement.

5.2.3.83 nvmlNvLinkInfo_v2_t

struct **nvmlNvLinkInfo_v2_t**
Struct to represent per device NVLINK information v2.

Public Members

unsigned int **version**
IN - the API version number.

unsigned int **isNvleEnabled**
OUT - NVLINK encryption enablement.

nvmlNvlinkFirmwareInfo_t **firmwareInfo**
OUT - NVLINK Firmware info.

5.2.3.84 nvmlNvLinkInfo_v3_t

struct **nvmlNvLinkInfo_v3_t**
Struct to represent per device NVLINK information v3.

Public Members

unsigned int **isNvleEnabled**
OUT - NVLINK encryption enablement.

nvmlNvlinkFirmwareInfo_t **firmwareInfo**
OUT - NVLINK Firmware info.

unsigned int **isNvleReady**
OUT - Whether NVLE is ready on the GPU.

5.2.3.85 nvmlNvLinkPowerThres_t

struct **nvmlNvLinkPowerThres_t**

Public Members

unsigned int **lowPwrThreshold**
Low power threshold.

5.2.3.86 nvmlNvLinkUtilizationControl_t

struct **nvmlNvLinkUtilizationControl_t**

Struct to define the NVLINK counter controls.

Public Members

nvmlNvLinkUtilizationCountUnits_t **units**

nvmlNvLinkUtilizationCountPktTypes_t **pktfilter**

5.2.3.87 nvmlNvlinkFirmwareInfo_t

struct **nvmlNvlinkFirmwareInfo_t**

Struct to represent NVLINK firmware information.

Public Members

nvmlNvlinkFirmwareVersion_t **firmwareVersion**[100]

OUT - NVLINK firmware version.

unsigned int **numValidEntries**

OUT - Number of valid firmware entries.

5.2.3.88 nvmlNvlinkFirmwareVersion_t

struct **nvmlNvlinkFirmwareVersion_t**

Struct to represent NVLINK firmware Semantic versioning and ucode type.

Public Members

unsigned char **ucodeType**

unsigned int **major**

unsigned int **minor**

unsigned int **subMinor**

5.2.3.89 nvmlNvlinkGetBwMode_v1_t

struct **nvmlNvlinkGetBwMode_v1_t**

Public Members

unsigned int **version**

unsigned int **bIsBest**

unsigned char **bwMode**

5.2.3.90 nvmlNvlinkGetRemapTableInfo_v1_t

struct **nvmlNvlinkGetRemapTableInfo_v1_t**

Structure for NVLink remap table information.

Public Members

unsigned int **alid**

ALID of the GPU.

unsigned int **flaRemapTabAddr**[4096]

FLA remap table addrs for all the remap table entries.

unsigned int **gpaRemapTabAddr**[4096]

SPA/GPA remap table addrs for all the remap table entries.

unsigned int **remapTabSize**

Number of entries in the remap tables.

5.2.3.91 nvmlNvlinkLockRemapTable_v1_t

struct **nvmlNvlinkLockRemapTable_v1_t**

Structure for the NVLink remap table and MSE lock operation.

Public Members

unsigned char **bRemapTableLocked**

Whether the remap table of the GPU is locked.

unsigned char **bRemapTableLockDisabled**

Whether locking of the remap table is disabled.

5.2.3.92 nvmlNvlinkNvlePrivateInfo_v1_t

struct **nvmlNvlinkNvlePrivateInfo_v1_t**

Structure for NVLink NVLE private (opaque) information.

Public Members

unsigned char **data**[256]

Buffer that receives the opaque private info blob.

unsigned int **size**

Number of valid bytes written to *data* on output.

5.2.3.93 nvmlNvlinkSetBwModeAsync_v1_t

struct **nvmlNvlinkSetBwModeAsync_v1_t**

Describes the parameters involved to setting Nvlink RBM mode asynchronously.

This structure holds the parameters needed to correctly set a Device's Nvlink Reduced Bandwidth Mode asynchronously. Polling for NVML_GPU_FABRIC_STATE_COMPLETED from *nvmlDeviceGetGpuFabricInfoV()* is needed to check if the setting was applied.

Public Members

unsigned int **bSetBest**

[in] - Set to the best available Bandwidth mode

unsigned int **bwMode**

[in] - Requested Bandwidth mode to set. Values can be found from *nvmlDeviceGetNvlinkSupportedBwModes()*

unsigned int **asyncPollTimeoutMs**

[out] - Time in ms to poll to validate bandwidth setting.

5.2.3.94 nvmlNvlinkSetBwMode_v1_t

struct **nvmlNvlinkSetBwMode_v1_t**

Public Members

unsigned int **version**

unsigned int **bSetBest**

unsigned char **bwMode**

5.2.3.95 nvmlNvlinkSetNvleReady_v1_t

struct **nvmlNvlinkSetNvleReady_v1_t**

Structure for the NVLink NVLE ready state.

Public Members

unsigned char **bNvleReady**
Whether NVLE is ready on the GPU.

5.2.3.96 nvmlNvlinkSetupNvleEncryptionKey_v1_t

struct **nvmlNvlinkSetupNvleEncryptionKey_v1_t**
Structure for NVLink NVLE encryption key setup.

Public Members

unsigned char **nvleKey**[32]
Key secret from which keyMgr will derive the key.

unsigned int **localGpuAlid**
ALID of the local GPU.

unsigned int **remoteGpuAlid**
ALID of the remote GPU.

nvmlNvlinkNvlePrivateInfo_v1_t **localGpuNvlePrivateInfo**
NVLE private info of the local GPU.

nvmlNvlinkNvlePrivateInfo_v1_t **remoteGpuNvlePrivateInfo**
NVLE private info of the remote GPU.

5.2.3.97 nvmlNvlinkSupportedBwModes_v1_t

struct **nvmlNvlinkSupportedBwModes_v1_t**

Public Members

unsigned int **version**

unsigned char **bwModes**[23]

unsigned char **totalBwModes**

5.2.3.98 nvmlNvlinkTelemetrySample_v1_t

struct **nvmlNvlinkTelemetrySample_v1_t**
Struct representing one (link, metric) telemetry request / response slot.

Public Members

unsigned int **linkId**

[in] LinkId

unsigned int **sampleType**

[in] Type of telemetry to sample, specified by `nvmlNvlinkTelemetrySampleType_t`

unsigned int **sampleCount**

[in,out]: Number of samples users need to allocate. If set to 0, will return max supported count of samples without touching the `samples` pointer.

unsigned long long ***samples**

[in,out]: Array of samples allocated by the user. Can be set to NULL when getting count

nvmlReturn_t **nvmlReturn**

[out]: Return code for retrieving this sample. This must be checked by the client before looking at any output values, as they are invalid if `nvmlReturn != NVML_SUCCESS`.

5.2.3.99 nvmlNvlinkTelemetrySamples_v1_t

struct **nvmlNvlinkTelemetrySamples_v1_t**

Batched NVLink telemetry request.

Public Members

unsigned int **telemetryCount**

[in] Number of valid entries in *telemetrySamples*

nvmlNvlinkTelemetrySample_v1_t ***telemetrySamples**

[in,out] Caller-allocated array of *telemetryCount* request slots

5.2.3.100 nvmlObservedMetrics_t

struct **nvmlObservedMetrics_t**

Observed metrics from the power model, including initial DRAMCLK estimates and current measurements.

Public Members

nvmlPwrModelMetricsDlppm1xDramclkEstimates_t **initialDramclkEst**[3]

Initial DRAMCLK estimates for different scenarios.

unsigned char **bValid**

Validity flag: non-zero if observed metrics are valid.

nvmlCoreRailMetrics_t **coreRail**
Observed core rail metrics.

nvmlRailMetrics_t **fbRail**
Observed fb rail metrics.

nvmlPmgrPwrTuple_t **tgpPwrTuple**
Observed Total Graphics Power (TGP) in milliwatts.

nvmlPwrModelMetricsDlppm1xPerf_t **perfMetrics**
Observed performance metrics.

5.2.3.101 nvmlPRMCounterInput_v1_t

struct **nvmlPRMCounterInput_v1_t**
PRM counter input values.

Public Members

unsigned int **localPort**
Local port number.

5.2.3.102 nvmlPRMCounterList_v1_t

struct **nvmlPRMCounterList_v1_t**
PRM Counter List Structure v1.

Public Members

unsigned int **numCounters**
Number of counters.

nvmlPRMCounter_v1_t ***counters**
Pointer to array of PRM counters.

5.2.3.103 nvmlPRMCounterValue_v1_t

struct **nvmlPRMCounterValue_v1_t**
PRM Counter Value Structure.

Public Members

nvmlReturn_t **status**
Status of the PRM counter read.

nvmlValueType_t **outputType**
Output value type.

nvmlValue_t **outputValue**
Output value.

5.2.3.104 nvmlPRMCounter_v1_t

struct **nvmlPRMCounter_v1_t**
PRM Counter Structure v1.

Public Members

unsigned int **counterId**
Counter ID, one of *nvmlPRMCounterId_t*.

nvmlPRMCounterInput_v1_t **inData**
PRM input values.

nvmlPRMCounterValue_v1_t **counterValue**
Counter value.

5.2.3.105 nvmlPRMTLV_v1_t

struct **nvmlPRMTLV_v1_t**
Main PRM input structure.

Public Members

unsigned **dataSize**
Size of the input TLV data.

unsigned **status**
OUT: status of the PRM command.

unsigned char **inData**[496]
IN: Input data in TLV format.

unsigned char **outData**[496]
OUT: Output PRM data in TLV format.

union *nvmlPRMTLV_v1_t*::[anonymous] **[anonymous]**

5.2.3.106 nvmlPSUInfo_t

struct **nvmlPSUInfo_t**

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

Public Members

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)

5.2.3.107 nvmlPciInfoExt_v1_t

struct **nvmlPciInfoExt_v1_t**

PCI information about a GPU device.

Public Members

unsigned int **version**

The version number of this struct.

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

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.

unsigned int **baseClass**
The 8-bit PCI base class code.

unsigned int **subClass**
The 8-bit PCI sub class code.

char **busId**[32]
The tuple domain:bus:device.function PCI identifier (& NULL terminator)

5.2.3.108 nvmlPciInfo_t

struct **nvmlPciInfo_t**
PCI information about a GPU device.

Public Members

char **busIdLegacy**[16]
The legacy tuple domain:bus:device.function PCI identifier (& NULL terminator)

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

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.

char **busId**[32]
The tuple domain:bus:device.function PCI identifier (& NULL terminator)

5.2.3.109 nvmlPdi_v1_t

struct **nvmlPdi_v1_t**
Struct to represent the NVML PDI information.

Public Members

unsigned int **version**
API version number.

unsigned long long **value**
64-bit PDI value

5.2.3.110 nvmlPerfMetricControllerSample_t

struct **nvmlPerfMetricControllerSample_t**
Performance metrics sample from a controller, which can be either DLPPC 2x or PFPP 1x.

Public Members

unsigned int **controllerType**
Controller type: NVML_PERF_METRICS_CONTROLLER_TYPE_DLPPC_2X or NVML_PERF_METRICS_CONTROLLER_TYPE_PFPP_1X.

nvmlPerfMetricsDlppc2xSample_t **dlppc2x**
DLPPC 2x controller sample data.

nvmlPerfMetricsPfpp1xSample_t **pfpp1x**
PFPP 1x controller sample data.

union *nvmlPerfMetricControllerSample_t*::[anonymous] **data**
Union containing controller-specific data.

5.2.3.111 nvmlPerfMetricsDlppc2xSample_t

struct **nvmlPerfMetricsDlppc2xSample_t**
Performance metrics sample for DLPPC 2x controller.

Public Members

nvmlObservedMetrics_t **observedMetrics**
Observed metrics from the DLPPC 2x controller.

5.2.3.112 nvmlPerfMetricsPfpp1xSample_t

struct **nvmlPerfMetricsPfpp1xSample_t**
Performance metrics sample for PFPP 1x controller.

Public Members

nvmlPwrModelMetricsPfpp1x_t **estimatedMetrics**
Estimated metrics from the PFPP 1x controller.

5.2.3.113 nvmlPerfMetricsSample_t

struct **nvmlPerfMetricsSample_t**
Single performance metrics sample containing data from one or more controllers.

Public Members

unsigned char **numControllerData**
Number of valid controller samples in this sample.

nvmlPerfMetricControllerSample_t **controllerData**[4]
Array of controller samples.

5.2.3.114 nvmlPerfMetricsSamples_v1_t

struct **nvmlPerfMetricsSamples_v1_t**
Collection of performance metrics samples (version 1).

This structure contains multiple samples for performance monitoring and profiling.

Public Members

unsigned int **numSamples**
Number of samples in the samples array.

nvmlPerfMetricsSample_t **samples**[13]
Array of performance metrics samples.

5.2.3.115 nvmlPlatformInfo_v1_t

struct **nvmlPlatformInfo_v1_t**

Structure to store platform information.

Deprecated:

The *nvmlPlatformInfo_v1_t* will be deprecated in the subsequent releases. Use *nvmlPlatformInfo_v2_t*

Public Members

unsigned int **version**

the API version number

unsigned char **ibGuid**[16]

Infiniband GUID reported by platform (for Blackwell, ibGuid is 8 bytes so indices 8-15 are zero)

unsigned char **rackGuid**[16]

GUID of the rack containing this GPU (for Blackwell rackGuid is 13 bytes so indices 13-15 are zero)

unsigned char **chassisPhysicalSlotNumber**

The slot number in the rack containing this GPU (includes switches)

unsigned char **computeSlotIndex**

The index within the compute slots in the rack containing this GPU (does not include switches)

unsigned char **nodeIndex**

Index of the node within the slot containing this GPU.

unsigned char **peerType**

Platform indicated NVLink-peer type (e.g. switch present or not)

unsigned char **moduleId**

ID of this GPU within the node.

5.2.3.116 nvmlPlatformInfo_v2_t

struct **nvmlPlatformInfo_v2_t**

Structure to store platform information (v2)

Public Members

unsigned int **version**

the API version number

unsigned char **ibGuid**[16]

Infiniband GUID reported by platform (for Blackwell, ibGuid is 8 bytes so indices 8-15 are zero)

unsigned char **chassisSerialNumber**[16]

Serial number of the chassis containing this GPU (for Blackwell it is 13 bytes so indices 13-15 are zero)

unsigned char **slotNumber**

The slot number in the chassis containing this GPU (includes switches)

unsigned char **trayIndex**

The tray index within the compute slots in the chassis containing this GPU (does not include switches)

unsigned char **hostId**

Index of the node within the slot containing this GPU.

unsigned char **peerType**

Platform indicated NVLink-peer type (e.g. switch present or not)

unsigned char **moduleId**

ID of this GPU within the node.

5.2.3.117 nvmlPmgrPwrTuple_t

struct **nvmlPmgrPwrTuple_t**

Power tuple containing power consumption in milliwatts.

Public Members

unsigned int **pwrmW**

Power consumption in milliwatts.

5.2.3.118 nvmlPowerSmoothingProfile_v1_t

struct **nvmlPowerSmoothingProfile_v1_t**

Power Smoothing Structure for Profile information.

Public Members

unsigned int **version**
: the API version number

unsigned int **profileId**
: The requested profile ID.

unsigned int **paramId**
: The requested parameter ID.

double **value**
: The requested value for the given parameter.

5.2.3.119 nvmlPowerSmoothingState_v1_t

struct **nvmlPowerSmoothingState_v1_t**
: Power Smoothing Structure for Feature Enablement.

Public Members

unsigned int **version**
: the API version number

nvmlEnableState_t **state**
: 0/Disabled or 1/Enabled

5.2.3.120 nvmlPowerValue_v2_t

struct **nvmlPowerValue_v2_t**
: Contains the power management limit.

Public Members

unsigned int **version**
: Structure format version (must be 1)

nvmlPowerScopeType_t **powerScope**
: [in] Device type: GPU or Total Module

unsigned int **powerValueMw**
: [out] Power value to retrieve or set in milliwatts

5.2.3.121 nvmlProcessDetailList_v1_t

struct **nvmlProcessDetailList_v1_t**

Information about all running processes on the GPU for the given mode.

Public Members

unsigned int **version**

Struct version, MUST be nvmlProcessDetailList_v1.

unsigned int **mode**

Process mode, One of *nvmlProcessMode_t*.

unsigned int **numProcArrayEntries**

Number of process entries in procArray.

nvmlProcessDetail_v1_t ***procArray**

Process array.

5.2.3.122 nvmlProcessDetail_v1_t

struct **nvmlProcessDetail_v1_t**

Information about running process on the GPU with protected memory.

Public Members

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.

unsigned int **gpuInstanceId**

If MIG is enabled, stores a valid GPU instance ID. gpuInstanceId is.

unsigned int **computeInstanceId**

If MIG is enabled, stores a valid compute instance ID. computeInstanceId.

unsigned long long **usedGpuCcProtectedMemory**

Amount of used GPU conf compute protected memory in bytes.

5.2.3.123 nvmlProcessInfo_t

struct **nvmlProcessInfo_t**

Information about running compute processes on the GPU.

Public Members

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.

unsigned int **gpuInstanceId**

If MIG is enabled, stores a valid GPU instance ID. gpuInstanceId is set to.

unsigned int **computeInstanceId**

If MIG is enabled, stores a valid compute instance ID. computeInstanceId is set to.

5.2.3.124 nvmlProcessInfo_v1_t

struct **nvmlProcessInfo_v1_t**

Information about running compute processes on the GPU, legacy version for older versions of the API.

Public Members

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.

5.2.3.125 nvmlProcessUtilizationInfo_v1_t

struct **nvmlProcessUtilizationInfo_v1_t**

Structure to store utilization value and process Id — version 1.

Public Members

unsigned long long **timeStamp**

CPU Timestamp in microseconds.

unsigned int **pid**

PID of process.

unsigned int **smUtil**

SM (3D/Compute) Util Value.

unsigned int **memUtil**
Frame Buffer Memory Util Value.

unsigned int **encUtil**
Encoder Util Value.

unsigned int **decUtil**
Decoder Util Value.

unsigned int **jpgUtil**
Jpeg Util Value.

unsigned int **ofaUtil**
Ofa Util Value.

5.2.3.126 nvmlProcessUtilizationSample_t

struct **nvmlProcessUtilizationSample_t**
Structure to store utilization value and process Id.

Public Members

unsigned int **pid**
PID of process.

unsigned long long **timeStamp**
CPU Timestamp in microseconds.

unsigned int **smUtil**
SM (3D/Compute) Util Value.

unsigned int **memUtil**
Frame Buffer Memory Util Value.

unsigned int **encUtil**
Encoder Util Value.

unsigned int **decUtil**
Decoder Util Value.

5.2.3.127 nvmlProcessesUtilizationInfo_v1_t

struct **nvmlProcessesUtilizationInfo_v1_t**
Structure to store utilization and process ID for each running process — version 1.

Public Members

unsigned int **version**

The version number of this struct.

unsigned int **processSamplesCount**

Caller-supplied array size, and returns number of processes running.

unsigned long long **lastSeenTimeStamp**

Return only samples with timestamp greater than lastSeenTimeStamp.

nvmlProcessUtilizationInfo_v1_t ***procUtilArray**

The array (allocated by caller) of the utilization of GPU SM, framebuffer, video encoder, video decoder, JPEG, and OFA.

5.2.3.128 nvmlPwrModelMetricsDlppm1xDramclkEstimates_t

struct **nvmlPwrModelMetricsDlppm1xDramclkEstimates_t**

DRAMCLK estimates containing multiple estimated metrics for different DRAMCLK frequencies.

Public Members

nvmlPwrModelMetricsDlppm1x_t **estimatedMetrics**[8]

Array of estimated metrics for each inference loop.

unsigned char **numEstimatedMetrics**

Number of valid entries in estimatedMetrics array.

5.2.3.129 nvmlPwrModelMetricsDlppm1xPerf_t

struct **nvmlPwrModelMetricsDlppm1xPerf_t**

Performance metrics for DLPPM 1x power model.

Public Members

unsigned int **perfms**

Performance metric in milliseconds.

5.2.3.130 nvmlPwrModelMetricsDlppm1x_t

struct **nvmlPwrModelMetricsDlppm1x_t**

Complete power model metrics for DLPPM 1x, including rail metrics and TGP power.

Public Members

unsigned char **bValid**

Validity flag: non-zero if metrics are valid.

nvmlCoreRailMetrics_t **coreRail**

Core rail metrics.

nvmlRailMetrics_t **fbRail**

Fb rail metrics.

nvmlPmgrPwrTuple_t **tgpPwrTuple**

Total Graphics Power (TGP) in milliwatts.

nvmlPwrModelMetricsDlppm1xPerf_t **perfMetrics**

Performance metrics.

5.2.3.131 nvmlPwrModelMetricsPfpp1x_t

struct **nvmlPwrModelMetricsPfpp1x_t**

Complete power model metrics for PFPP 1x, including estimated metrics and key operating points.

Public Members

unsigned char **numVfPoints**

Number of valid vf points.

nvmlPwrModelMetricsSamplePfpp1x_t **estimatedMetrics**[32]

Array of estimated metrics for different operating points.

unsigned char **bValid**

Validity flag: non-zero if metrics are valid.

nvmlPwrModelOperatingPointPfpp1x_t **maxPerfPerWattPoint**

Operating point with maximum performance per watt.

nvmlPwrModelOperatingPointPfpp1x_t **fmaxAtVmaxPoint**

Operating point at maximum frequency and voltage.

unsigned int **tgpHeadroommW**

TGP headroom in milliwatts.

5.2.3.132 nvmlPwrModelMetricsSamplePfpp1x_t

struct **nvmlPwrModelMetricsSamplePfpp1x_t**

Power model metrics sample for PFPP 1x, containing frequency inputs and estimated TGP.

Public Members

unsigned int **freqkHz**[16]

Array of input frequencies in kilohertz for each domain.

unsigned int **estTgpPwrmW**

Estimated Total Graphics Power in milliwatts.

5.2.3.133 nvmlPwrModelOperatingPointPfpp1x_t

struct **nvmlPwrModelOperatingPointPfpp1x_t**

Operating point for PFPP 1x power model, defining a frequency-power pair.

Public Members

unsigned int **freqkHz**

Operating frequency in kilohertz.

unsigned int **pwrmW**

Power consumption at this frequency in milliwatts.

5.2.3.134 nvmlRailMetrics_t

struct **nvmlRailMetrics_t**

Metrics for a single power rail, including frequency and utilization.

Public Members

unsigned int **freqkHz**

Frequency in kilohertz.

unsigned long long **utilPct**

Utilization percentage (fixed-point)

5.2.3.135 nvmlRemappedRowsInfo_v2_t

struct **nvmlRemappedRowsInfo_v2_t**

Public Members

unsigned int **corrActiveRemaps**

Number of active row remappings due to correctable errors.

unsigned int **corrInactiveRemaps**

Number of inactive row remappings due to correctable errors.

unsigned int **uncActiveRemaps**

Number of active row remappings due to uncorrectable errors.

unsigned int **uncInactiveRemaps**

Number of inactive row remappings due to uncorrectable errors.

unsigned int **bPending**

Whether or not there is any pending row remapping; 0 indicates not pending, 1 indicates pending.

unsigned int **bFailureOccurred**

Whether or not there's any row remapping failure in the past; 0 indicates no failure, 1 indicates failure occurred.

5.2.3.136 nvmlRepairStatus_v1_t

struct **nvmlRepairStatus_v1_t**

Struct to represent the NVML repair status.

Public Members

unsigned int **version**

API version number.

unsigned int **bChannelRepairPending**

Reference to *unsigned* int.

unsigned int **bTpcRepairPending**

Reference to *unsigned* int.

5.2.3.137 nvmlRowRemapperHistogramValues_t

struct **nvmlRowRemapperHistogramValues_t**

Possible values that classify the remap availability for each bank.

The max field will contain the number of banks that have maximum remap availability (all reserved rows are available). None means that there are no reserved rows available.

Public Members

unsigned int **max**

unsigned int **high**

unsigned int **partial**

unsigned int **low**

unsigned int **none**

5.2.3.138 nvmlRusdSettings_v1_t

struct **nvmlRusdSettings_v1_t**

Stores RUSD polling configuration for a GPU.

Public Members

unsigned int **version**

unsigned long long **pollMask**

Bitmask of polling data. 0 value means the GPU's RUSD polling mask is cleared.

5.2.3.139 nvmlSample_t

struct **nvmlSample_t**

Information for Sample.

Public Members

unsigned long long **timeStamp**

CPU Timestamp in microseconds.

nvmlValue_t **sampleValue**

Sample Value.

5.2.3.140 nvmlSetMemoryLimits_v1_t

struct **nvmlSetMemoryLimits_v1_t**

Describes the memory limits that can be set for the device.

This structure holds the necessary information that can be used to set the soft and hard memory limits of a device.

The softLimit is the amount of memory that is guaranteed before allocations may fail due to memory pressure.

The hardLimit is the maximum amount of memory that can be allocated.

This should be cross-referenced with *nvmlDeviceGetMemoryInfo_v2* while in that cgroup to see how much memory is actually available to allocate for the device.

To clear the limits, set softLimit to 0, and hardLimit to *NVML_DEVICE_MEMORY_LIMIT_MAX*. Removing the cgroup will also clear any limits.

Public Members

const char ***nameSpace**

[in] Full path to sysfs cgroup file name

unsigned long long **softLimit**

[in] Soft memory limit in Bytes.

unsigned long long **hardLimit**

[in] Hard memory limit in Bytes.

5.2.3.141 nvmlSystemConfComputeSettings_v1_t

struct **nvmlSystemConfComputeSettings_v1_t**

Confidential Compute System settings.

Public Members

unsigned int **version**

unsigned int **environment**

unsigned int **ccFeature**

unsigned int **devToolsMode**

unsigned int **multiGpuMode**

5.2.3.142 nvmlSystemConfComputeSettings_v2_t

struct **nvmlSystemConfComputeSettings_v2_t**

Confidential Compute System settings, version 2.

Adds *tdispState*, the TDISP state of the GPUs, reported as a single value so that callers do not have to combine flags.

Public Members

unsigned int **version**

unsigned int **environment**

unsigned int **ccFeature**

unsigned int **devToolsMode**

unsigned int **multiGpuMode**

unsigned int **tdispState**

5.2.3.143 nvmlSystemDriverBranchInfo_v1_t

struct **nvmlSystemDriverBranchInfo_v1_t**

Structure to store Driver branch information.

Public Members

unsigned int **version**

The version number of this struct.

char **branch**[80]

driver branch

5.2.3.144 nvmlSystemEventData_v1_t

struct **nvmlSystemEventData_v1_t**

nvmlSystemEventData_v1_t

Public Members

unsigned long long **eventType**

Information about what specific system event occurred.

unsigned int **gpuId**

gpuId in PCI format

5.2.3.145 nvmlSystemEventSetCreateRequest_v1_t

struct **nvmlSystemEventSetCreateRequest_v1_t**

nvmlSystemEventSetCreateRequest

Public Members

unsigned int **version**
the API version number

nvmlSystemEventSet_t **set**
system event set

5.2.3.146 nvmlSystemEventSetFreeRequest_v1_t

struct **nvmlSystemEventSetFreeRequest_v1_t**
nvmlSystemEventSetFreeRequest

Public Members

unsigned int **version**
the API version number

nvmlSystemEventSet_t **set**
system event set

5.2.3.147 nvmlSystemEventSetWaitRequest_v1_t

struct **nvmlSystemEventSetWaitRequest_v1_t**
nvmlSystemEventSetWait

Public Members

unsigned int **version**
input/output: the API version number

unsigned int **timeoutms**
input: time to sleep waiting for event. If timeoutms is zero, skip waiting for event.

nvmlSystemEventSet_t **set**
input: system event set

nvmlSystemEventData_v1_t ***data**
input/output: array of event data, owned by caller

unsigned int **dataSize**
input: the size of data array

unsigned int **numEvent**
output: number of event collected.

5.2.3.148 nvmlSystemRegisterEventRequest_v1_t

struct **nvmlSystemRegisterEventRequest_v1_t**

nvmlSystemRegisterEventRequest

Public Members

unsigned int **version**

the API version number

unsigned long long **eventTypes**

Bitmask of *Event Types* to record For example eventTypes = (nvmlEventTypeBind | nvmlEventTypeUnbind) to listen to both Bind and Unbind events.

nvmlSystemEventSet_t **set**

Set to which add new event types.

5.2.3.149 nvmlTemperature_v1_t

struct **nvmlTemperature_v1_t**

Structure used to encapsulate temperature info.

Public Members

unsigned int **version**

nvmlTemperatureSensors_t **sensorType**

int **temperature**

5.2.3.150 nvmlUUID_v1_t

struct **nvmlUUID_v1_t**

Struct to represent NVML UUID information.

Public Members

unsigned int **version**

API version number.

unsigned int **type**

One of `nvmlUUIDType_t`.

nvmlUUIDValue_t **value**

One of *`nvmlUUIDValue_t`*, to be set based on the UUID format.

5.2.3.151 nvmlUnitFanInfo_t

struct **nvmlUnitFanInfo_t**

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

Public Members

unsigned int **speed**

Fan speed (RPM)

nvmlFanState_t **state**

Flag that indicates whether fan is working properly.

5.2.3.152 nvmlUnitFanSpeeds_t

struct **nvmlUnitFanSpeeds_t**

Fan speed readings for an entire S-class unit.

Public Members

nvmlUnitFanInfo_t **fans**[24]

Fan speed data for each fan.

unsigned int **count**

Number of fans in unit.

5.2.3.153 nvmlUnitInfo_t

struct **nvmlUnitInfo_t**

Static S-class unit info.

Public Members

char **name**[96]

Product name.

char **id**[96]

Product identifier.

char **serial**[96]

Product serial number.

char **firmwareVersion**[96]

Firmware version.

5.2.3.154 nvmlUnrepairableMemoryStatus_v1_t

struct **nvmlUnrepairableMemoryStatus_v1_t**

Struct to represent the NVML unrepairable memory status.

Public Members

unsigned int **bUnrepairableMemory**

Reference to *unsigned* int.

5.2.3.155 nvmlUtilization_t

struct **nvmlUtilization_t**

Utilization information for a device.

Each sample period may be between 1 second and 1/6 second, depending on the product being queried.

Public Members

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.

5.2.3.156 nvmlVgpuCreatablePlacementInfo_v1_t

struct **nvmlVgpuCreatablePlacementInfo_v1_t**

Structure to store creatable vGPU placement information — version 1.

Public Members

unsigned int **version**

IN: The version number of this struct.

nvmlVgpuTypeId_t **vgpuTypeId**

IN: Handle to vGPU type.

unsigned int **count**

IN/OUT: Count of the placement IDs.

unsigned int ***placementIds**

IN/OUT: Placement IDs for the vGPU type.

unsigned int **placementSize**
OUT: The number of slots occupied by the vGPU type.

5.2.3.157 nvmlVgpuHeterogeneousMode_v1_t

struct **nvmlVgpuHeterogeneousMode_v1_t**
Structure to store the vGPU heterogeneous mode of device — version 1.

Public Members

unsigned int **version**
The version number of this struct.

unsigned int **mode**
The vGPU heterogeneous mode.

5.2.3.158 nvmlVgpuInstanceUtilizationInfo_v1_t

struct **nvmlVgpuInstanceUtilizationInfo_v1_t**
Structure to store Utilization Value and vgpuInstance Info — Version 1.

Public Members

unsigned long long **timeStamp**
CPU Timestamp in microseconds.

nvmlVgpuInstance_t **vgpuInstance**
vGPU Instance

nvmlValue_t **smUtil**
SM (3D/Compute) Util Value.

nvmlValue_t **memUtil**
Frame Buffer Memory Util Value.

nvmlValue_t **encUtil**
Encoder Util Value.

nvmlValue_t **decUtil**
Decoder Util Value.

nvmlValue_t **jpgUtil**
Jpeg Util Value.

nvmlValue_t **ofaUtil**
Ofa Util Value.

5.2.3.159 nvmlVgpuInstanceUtilizationSample_t

struct **nvmlVgpuInstanceUtilizationSample_t**

Structure to store Utilization Value and vgpuInstance.

Public Members

nvmlVgpuInstance_t **vgpuInstance**

vGPU Instance

unsigned long long **timeStamp**

CPU Timestamp in microseconds.

nvmlValue_t **smUtil**

SM (3D/Compute) Util Value.

nvmlValue_t **memUtil**

Frame Buffer Memory Util Value.

nvmlValue_t **encUtil**

Encoder Util Value.

nvmlValue_t **decUtil**

Decoder Util Value.

5.2.3.160 nvmlVgpuInstancesUtilizationInfo_v1_t

struct **nvmlVgpuInstancesUtilizationInfo_v1_t**

Structure to store recent utilization for vGPU instances running on a device — version 1.

Public Members

unsigned int **version**

The version number of this struct.

nvmlValueType_t **sampleValType**

Hold the type of returned sample values.

unsigned int **vgpuInstanceCount**

Hold the number of vGPU instances.

unsigned long long **lastSeenTimeStamp**

Return only samples with timestamp greater than lastSeenTimeStamp.

nvmlVgpuInstanceUtilizationInfo_v1_t ***vgpuUtilArray**

The array (allocated by caller) in which vGPU utilization are returned.

5.2.3.161 nvmlVgpuLicenseExpiry_t

struct **nvmlVgpuLicenseExpiry_t**

Structure to store the vGPU license expiry details.

Public Members

unsigned int **year**

Year of license expiry.

unsigned short **month**

Month of license expiry.

unsigned short **day**

Day of license expiry.

unsigned short **hour**

Hour of license expiry.

unsigned short **min**

Minutes of license expiry.

unsigned short **sec**

Seconds of license expiry.

unsigned char **status**

License expiry status.

5.2.3.162 nvmlVgpuLicenseInfo_t

struct **nvmlVgpuLicenseInfo_t**

Public Members

unsigned char **isLicensed**

License status.

nvmlVgpuLicenseExpiry_t **licenseExpiry**

License expiry information.

unsigned int **currentState**

Current license state.

5.2.3.163 nvmlVgpuMetadata_t

struct **nvmlVgpuMetadata_t**

vGPU metadata structure.

Public Members

unsigned int **version**

Current version of the structure.

unsigned int **revision**

Current revision of the structure.

nvmlVgpuGuestInfoState_t **guestInfoState**

Current state of Guest-dependent fields.

char **guestDriverVersion**[80]

Version of driver installed in guest.

char **hostDriverVersion**[80]

Version of driver installed in host.

unsigned int **reserved**[6]

Reserved for internal use.

unsigned int **vgpuVirtualizationCaps**

vGPU virtualization capabilities bitfield

unsigned int **guestVgpuVersion**

vGPU version of guest driver

unsigned int **opaqueDataSize**

Size of opaque data field in bytes.

char **opaqueData**[4]

Opaque data.

5.2.3.164 nvmlVgpuPgpuCompatibility_t

struct **nvmlVgpuPgpuCompatibility_t**

vGPU-pGPU compatibility structure

Public Members

nvmlVgpuVmCompatibility_t **vgpuVmCompatibility**
Compatibility of vGPU VM. See *nvmlVgpuVmCompatibility_t*.

nvmlVgpuPgpuCompatibilityLimitCode_t **compatibilityLimitCode**
Limiting factor for vGPU-pGPU compatibility. See *nvmlVgpuPgpuCompatibilityLimitCode_t*.

5.2.3.165 nvmlVgpuPgpuMetadata_t

struct **nvmlVgpuPgpuMetadata_t**
Physical GPU metadata structure.

Public Members

unsigned int **version**
Current version of the structure.

unsigned int **revision**
Current revision of the structure.

char **hostDriverVersion**[80]
Host driver version.

unsigned int **pgpuVirtualizationCaps**
Pgpu virtualization capabilities bitfield.

unsigned int **reserved**[5]
Reserved for internal use.

nvmlVgpuVersion_t **hostSupportedVgpuRange**
vGPU version range supported by host driver

unsigned int **opaqueDataSize**
Size of opaque data field in bytes.

char **opaqueData**[4]
Opaque data.

5.2.3.166 nvmlVgpuPlacementId_v1_t

struct **nvmlVgpuPlacementId_v1_t**
Structure to store the placement ID of vGPU instance — version 1.

Public Members

unsigned int **version**
: The version number of this struct.

unsigned int **placementId**
: Placement ID of the active vGPU instance.

5.2.3.167 nvmlVgpuPlacementList_v1_t

struct **nvmlVgpuPlacementList_v1_t**
: Structure to store the list of vGPU placements — version 1.

Public Members

unsigned int **version**
: The version number of this struct.

unsigned int **placementSize**
: The number of slots occupied by the vGPU type.

unsigned int **count**
: Count of placement IDs fetched.

unsigned int ***placementIds**
: Placement IDs for the vGPU type.

5.2.3.168 nvmlVgpuPlacementList_v2_t

struct **nvmlVgpuPlacementList_v2_t**
: Structure to store the list of vGPU placements — version 2.

Public Members

unsigned int **version**
: IN: The version number of this struct.

unsigned int **placementSize**
: OUT: The number of slots occupied by the vGPU type.

unsigned int **count**
: IN/OUT: Count of the placement IDs.

unsigned int ***placementIds**
: IN/OUT: Placement IDs for the vGPU type.

unsigned int **mode**

IN: The vGPU mode. Either NVML_VGPU_PGPU_HETEROGENEOUS_MODE or NVML_VGPU_PGPU_HOMOGENEOUS_MODE.

5.2.3.169 nvmlVgpuProcessUtilizationInfo_v1_t

struct **nvmlVgpuProcessUtilizationInfo_v1_t**

Structure to store Utilization Value, vgpuInstance and subprocess information for process running on vGPU instance — version 1.

Public Members

char **processName**[64]

Name of process running within the vGPU VM.

unsigned long long **timeStamp**

CPU Timestamp in microseconds.

nvmlVgpuInstance_t **vgpuInstance**

vGPU Instance

unsigned int **pid**

PID of process running within the vGPU VM.

unsigned int **smUtil**

SM (3D/Compute) Util Value.

unsigned int **memUtil**

Frame Buffer Memory Util Value.

unsigned int **encUtil**

Encoder Util Value.

unsigned int **decUtil**

Decoder Util Value.

unsigned int **jpgUtil**

Jpeg Util Value.

unsigned int **ofaUtil**

Ofa Util Value.

5.2.3.170 nvmlVgpuProcessUtilizationSample_t

struct **nvmlVgpuProcessUtilizationSample_t**

Structure to store Utilization Value, vgpuInstance and subprocess information.

Public Members

nvmlVgpuInstance_t **vgpuInstance**

vGPU Instance

unsigned int **pid**

PID of process running within the vGPU VM.

char **processName**[64]

Name of process running within the vGPU VM.

unsigned long long **timeStamp**

CPU Timestamp in microseconds.

unsigned int **smUtil**

SM (3D/Compute) Util Value.

unsigned int **memUtil**

Frame Buffer Memory Util Value.

unsigned int **encUtil**

Encoder Util Value.

unsigned int **decUtil**

Decoder Util Value.

5.2.3.171 nvmlVgpuProcessesUtilizationInfo_v1_t

struct **nvmlVgpuProcessesUtilizationInfo_v1_t**

Structure to store recent utilization, vgpuInstance and subprocess information for processes running on vGPU instances active on a device — version 1.

Public Members

unsigned int **version**

The version number of this struct.

unsigned int **vgpuProcessCount**

Hold the number of processes running on vGPU instances.

unsigned long long **lastSeenTimeStamp**

Return only samples with timestamp greater than lastSeenTimeStamp.

nvmlVgpuProcessUtilizationInfo_v1_t ***vgpuProcUtilArray**

The array (allocated by caller) in which utilization of processes running on vGPU instances are returned.

5.2.3.172 nvmlVgpuRuntimeState_v1_t

struct **nvmlVgpuRuntimeState_v1_t**

Structure to store the information of vGPU runtime state — version 1.

Public Members

unsigned int **version**

IN: The version number of this struct.

unsigned long long **size**

OUT: The runtime state size of the vGPU instance.

5.2.3.173 nvmlVgpuSchedulerCapabilities_t

struct **nvmlVgpuSchedulerCapabilities_t**

Structure to store the vGPU scheduler capabilities.

Public Members

unsigned int **supportedSchedulers**[3]

List the supported vGPU schedulers on the device.

unsigned int **maxTimeslice**

Maximum timeslice value in ns.

unsigned int **minTimeslice**

Minimum timeslice value in ns.

unsigned int **isArrModeSupported**

Flag to check Adaptive Round Robin mode enabled/disabled.

unsigned int **maxFrequencyForARR**

Maximum frequency for Adaptive Round Robin mode.

unsigned int **minFrequencyForARR**

Minimum frequency for Adaptive Round Robin mode.

unsigned int **maxAvgFactorForARR**
Maximum averaging factor for Adaptive Round Robin mode.

unsigned int **minAvgFactorForARR**
Minimum averaging factor for Adaptive Round Robin mode.

5.2.3.174 nvmlVgpuSchedulerGetState_t

struct **nvmlVgpuSchedulerGetState_t**
Structure to store the vGPU scheduler state.

Public Members

unsigned int **schedulerPolicy**
Scheduler policy.

unsigned int **arrMode**
Adaptive Round Robin scheduler mode. One of the NVML_VGPU_SCHEDULER_ARR_*.

nvmlVgpuSchedulerParams_t **schedulerParams**

5.2.3.175 nvmlVgpuSchedulerLogEntry_t

struct **nvmlVgpuSchedulerLogEntry_t**
Structure to store the state and logs of a software runlist.

Public Members

unsigned long long **timestamp**
Timestamp in ns when this software runlist was preeempted.

unsigned long long **timeRunTotal**
Total time in ns this software runlist has run.

unsigned long long **timeRun**
Time in ns this software runlist ran before preemption.

unsigned int **swRunlistId**
Software runlist Id.

unsigned long long **targetTimeSlice**
The actual timeslice after deduction.

unsigned long long **cumulativePreemptionTime**
Preemption time in ns for this SW runlist.

5.2.3.176 nvmlVgpuSchedulerLogEntry_v2_t

struct **nvmlVgpuSchedulerLogEntry_v2_t**

Structure to store the state and logs of a software runlist.

Public Members

unsigned long long **timestamp**

OUT: Timestamp in ns when this software runlist was preeempted.

unsigned long long **timeRunTotal**

OUT: Total time in ns this software runlist has run.

unsigned long long **timeRun**

OUT: Time in ns this software runlist ran before preemption.

unsigned int **swRunlistId**

OUT: Software runlist Id.

unsigned long long **targetTimeSlice**

OUT: The actual timeslice after deduction.

unsigned long long **cumulativePreemptionTime**

OUT: Preemption time in ns for this SW runlist.

unsigned int **weight**

OUT: Current weight of this SW runlist.

5.2.3.177 nvmlVgpuSchedulerLogInfo_v1_t

struct **nvmlVgpuSchedulerLogInfo_v1_t**

Structure to store vGPU scheduler log information — Version 1.

Public Members

unsigned int **version**

IN: The version number of this struct.

unsigned int **engineId**

IN: Engine whose software runlist log entries are fetched. One of One of NVML_VGPU_SCHEDULER_ENGINE_TYPE_*.

unsigned int **schedulerPolicy**

OUT: Scheduler policy.

unsigned int **arrMode**

OUT: Adaptive Round Robin scheduler mode. One of the NVML_VGPU_SCHEDULER_ARR_*.

nvmlVgpuSchedulerParams_t **schedulerParams**

OUT: vGPU Scheduler Parameters.

unsigned int **entriesCount**

OUT: Count of log entries fetched.

nvmlVgpuSchedulerLogEntry_t **logEntries**[200]

OUT: Structure to store the state and logs of a software runlist.

5.2.3.178 nvmlVgpuSchedulerLogInfo_v2_t

struct **nvmlVgpuSchedulerLogInfo_v2_t**

Structure to store vGPU scheduler log information.

Public Members

unsigned int **engineId**

IN: Engine whose software runlist log entries are fetched. One of One of NVML_VGPU_SCHEDULER_ENGINE_TYPE_*.

unsigned int **schedulerPolicy**

OUT: Scheduler policy.

unsigned int **avgFactor**

OUT: Average factor in compensating the timeslice for Adaptive Round Robin mode. 0 when there is no active scheduling.

unsigned int **timeslice**

OUT: The timeslice in ns for each software run list as configured, or the default value otherwise. 0 when there is no active scheduling.

unsigned int **entriesCount**

OUT: Count of log entries fetched.

nvmlVgpuSchedulerLogEntry_v2_t **logEntries**[200]

OUT: Structure to store the state and logs of a software runlist.

5.2.3.179 nvmlVgpuSchedulerLog_t

struct **nvmlVgpuSchedulerLog_t**

Structure to store a vGPU software scheduler log.

Public Members

unsigned int **engineId**
Engine whose software runlist log entries are fetched.

unsigned int **schedulerPolicy**
Scheduler policy.

unsigned int **arrMode**
Adaptive Round Robin scheduler mode. One of the NVML_VGPU_SCHEDULER_ARR_*.

nvmlVgpuSchedulerParams_t **schedulerParams**

unsigned int **entriesCount**
Count of log entries fetched.

nvmlVgpuSchedulerLogEntry_t **logEntries**[200]

5.2.3.180 nvmlVgpuSchedulerSetState_t

struct **nvmlVgpuSchedulerSetState_t**
Structure to set the vGPU scheduler state.

Public Members

unsigned int **schedulerPolicy**
Scheduler policy.

unsigned int **enableARRMode**
Adaptive Round Robin scheduler.

nvmlVgpuSchedulerSetParams_t **schedulerParams**

5.2.3.181 nvmlVgpuSchedulerStateInfo_v1_t

struct **nvmlVgpuSchedulerStateInfo_v1_t**
Structure to store vGPU scheduler state information — Version 1.

Public Members

unsigned int **version**
IN: The version number of this struct.

unsigned int **engineId**

IN: Engine whose software scheduler state info is fetched. One of NVML_VGPU_SCHEDULER_ENGINE_TYPE_*.

unsigned int **schedulerPolicy**

OUT: Scheduler policy.

unsigned int **arrMode**

OUT: Adaptive Round Robin scheduler mode. One of the NVML_VGPU_SCHEDULER_ARR_*.

nvmlVgpuSchedulerParams_t **schedulerParams**

OUT: vGPU Scheduler Parameters.

5.2.3.182 nvmlVgpuSchedulerStateInfo_v2_t

struct **nvmlVgpuSchedulerStateInfo_v2_t**

Structure to store vGPU scheduler state information.

Public Members

unsigned int **engineId**

IN: Engine whose software scheduler state info is fetched. One of NVML_VGPU_SCHEDULER_ENGINE_TYPE_*.

unsigned int **schedulerPolicy**

OUT: Scheduler policy.

unsigned int **avgFactor**

OUT: Average factor in compensating the timeslice for Adaptive Round Robin mode. 0 when there is no active scheduling.

unsigned int **timeslice**

OUT: The timeslice in ns for each software run list as configured, or the default value otherwise. 0 when there is no active scheduling.

5.2.3.183 nvmlVgpuSchedulerState_v1_t

struct **nvmlVgpuSchedulerState_v1_t**

Structure to set vGPU scheduler state information — version 1.

Public Members

unsigned int **version**

IN: The version number of this struct.

unsigned int **engineId**

IN: One of NVML_VGPU_SCHEDULER_ENGINE_TYPE_*.

unsigned int **schedulerPolicy**

IN: Scheduler policy.

unsigned int **enableARRMode**

IN: Adaptive Round Robin scheduler.

nvmlVgpuSchedulerSetParams_t **schedulerParams**

IN: vGPU Scheduler Parameters.

5.2.3.184 nvmlVgpuSchedulerState_v2_t

struct **nvmlVgpuSchedulerState_v2_t**

Structure to set vGPU scheduler state information.

Public Members

unsigned int **engineId**

IN: One of NVML_VGPU_SCHEDULER_ENGINE_TYPE_*.

unsigned int **schedulerPolicy**

IN: Scheduler policy.

unsigned int **avgFactor**

IN: Average factor in compensating the timeslice for Adaptive Round Robin mode. 0 or unspecified uses default.

unsigned int **frequency**

IN: Frequency for Adaptive Round Robin mode. 0 or unspecified uses default.

5.2.3.185 nvmlVgpuTypeBar1Info_v1_t

struct **nvmlVgpuTypeBar1Info_v1_t**

Structure to store BAR1 size information of vGPU type — Version 1.

Public Members

unsigned int **version**

The version number of this struct.

unsigned long long **bar1Size**

BAR1 size in megabytes.

5.2.3.186 nvmlVgpuTypeIdInfo_v1_t

struct **nvmlVgpuTypeIdInfo_v1_t**

Structure to store the vGPU type IDs — version 1.

Public Members

unsigned int **version**

IN: The version number of this struct.

unsigned int **vgpuCount**

IN/OUT: Number of vGPU types.

nvmlVgpuTypeId_t ***vgpuTypeIds**

OUT: List of vGPU type IDs.

5.2.3.187 nvmlVgpuTypeMaxInstance_v1_t

struct **nvmlVgpuTypeMaxInstance_v1_t**

Structure to store the maximum number of possible vGPU type IDs — version 1.

Public Members

unsigned int **version**

IN: The version number of this struct.

nvmlVgpuTypeId_t **vgpuTypeId**

IN: Handle to vGPU type.

unsigned int **maxInstancePerGI**

OUT: Maximum number of vGPU instances per GPU instance.

5.2.3.188 nvmlVgpuVersion_t

struct **nvmlVgpuVersion_t**

Structure representing range of vGPU versions.

Public Members

unsigned int **minVersion**

Minimum vGPU version.

unsigned int **maxVersion**

Maximum vGPU version.

5.2.3.189 nvmlViolationTime_t

struct **nvmlViolationTime_t**

Struct to hold perf policy violation status data.

Public Members

unsigned long long **referenceTime**

referenceTime represents CPU timestamp in microseconds

unsigned long long **violationTime**

violationTime in Nanoseconds

5.2.3.190 nvmlWorkloadPowerProfileCurrentProfiles_v1_t

struct **nvmlWorkloadPowerProfileCurrentProfiles_v1_t**

Current Profiles.

Public Members

unsigned int **version**

nvmlMask255_t **perfProfilesMask**

Mask bit set to true for each valid performance profile.

nvmlMask255_t **requestedProfilesMask**

Mask of currently requested performance profiles.

nvmlMask255_t **enforcedProfilesMask**

Mask of currently enforced performance profiles post all arbitrations among the requested profiles.

5.2.3.191 nvmlWorkloadPowerProfileInfo_v1_t

struct **nvmlWorkloadPowerProfileInfo_v1_t**

Profile Metadata.

Public Members

unsigned int **version**

the API version number

unsigned int **profileId**

Performance Profile Id to provide semantic name such as compute, Memory, Max-Q...

unsigned int **priority**
Priority of the profile.

nvmlMask255_t **conflictingMask**
Mask of conflicting performance profiles.

5.2.3.192 nvmlWorkloadPowerProfileProfilesInfo_v1_t

struct **nvmlWorkloadPowerProfileProfilesInfo_v1_t**
Profiles Info.

Public Members

unsigned int **version**
the API version number

nvmlMask255_t **perfProfilesMask**
Mask bit set to true for each valid performance profile.

nvmlWorkloadPowerProfileInfo_t **perfProfile**[(255)]
Array of performance profile info parameters.

5.2.3.193 nvmlWorkloadPowerProfileRequestedProfiles_v1_t

struct **nvmlWorkloadPowerProfileRequestedProfiles_v1_t**
Requested Profiles.

Public Members

unsigned int **version**
the API version number

nvmlMask255_t **requestedProfilesMask**
Mask of 255 bits, each bit representing index of respective perf profile.

5.2.3.194 nvmlWorkloadPowerProfileUpdateProfiles_v1_t

struct **nvmlWorkloadPowerProfileUpdateProfiles_v1_t**
Update Profiles.

Public Members

nvmlPowerProfileOperation_t **operation**
Operation to perform.

nvmlMask255_t **updateProfilesMask**

Mask of 255 bits, each bit representing index of respective perf profile.

nvmlAccountingStats_t
Describes accounting statistics of a process.

nvmlAccountingStats_v2_t
Describes accounting statistics (v2) of a process.

nvmlActiveVgpuInstanceInfo_v1_t
Structure to store active vGPU instance information Version 1.

nvmlAdaptiveTgpModeInfo_v1_t
Stores Adaptive TGP Mode state and telemetry for a GPU.

nvmlBAR1Memory_t
BAR1 Memory allocation Information for a device.

nvmlBBXTimeData_v1_t
BBX Time Data.

nvmlBridgeChipHierarchy_t
This structure stores the complete Hierarchy of the Bridge Chip within the board.

nvmlBridgeChipInfo_t
Information about the Bridge Chip Firmware.

nvmlC2cModeInfo_v1_t
C2C Mode information for a device.

nvmlCPERCursor_v1_t
CPER query and cursor parameters.

nvmlClkMonFaultInfo_t
Clock Monitor error types.

nvmlClkMonStatus_t
Clock Monitor Status.

nvmlClockOffset_v1_t
Clock offset info.

nvmlComputeInstanceInfo_t

nvmlComputeInstancePlacement_t

nvmlComputeInstanceProfileInfo_t
Compute instance profile information.

nvmlComputeInstanceProfileInfo_v2_t
Compute instance profile information (v2).

nvmlComputeInstanceProfileInfo_v3_t
Compute instance profile information (v3).

nvmlConfComputeGetKeyRotationThresholdInfo_v1_t

nvmlConfComputeGpuAttestationReport_t

nvmlConfComputeGpuCertificate_t

nvmlConfComputeMemSizeInfo_t
Protected memory size.

nvmlConfComputeSetKeyRotationThresholdInfo_v1_t

nvmlConfComputeSystemCaps_t

nvmlConfComputeSystemState_t

nvmlCoolerInfo_v1_t

nvmlCoreRailMetrics_t
Metrics for all core rails in the system.

nvmlDeviceAddressingMode_v1_t
Struct to represent device addressing mode information.

nvmlDeviceAttributes_t

nvmlDeviceCapabilities_v1_t
Device capabilities.

nvmlDeviceCurrentClockFreqs_v1_t
Device current clocks string.

nvmlDevicePerfModes_v1_t
Device performance modes string.

nvmlDevicePowerMizerModes_v1_t

nvmlDramEncryptionInfo_v1_t
DRAM Encryption Info.

nvmlEccBankRemapperHistogram_v1_t
Structure to store bank remapper histogram.

nvmlEccBankRemapperStatus_v1_t
Structure to store bank remapper status.

nvmlEccErrorCounts_t
Detailed ECC error counts for a device.

nvmlEccSramErrorStatus_v1_t
Structure to store SRAM uncorrectable error counters.

nvmlEccSramUniqueUncorrectedErrorCounts_v1_t

nvmlEccSramUniqueUncorrectedErrorEntry_v1_t

nvmlEncoderSessionInfo_t
Structure to hold encoder session data.

nvmlEventData_t
Information about occurred event.

nvmlEventSetGetContextCount_v1_t
Parameters for retrieving the number of context records associated with the most recent event.

nvmlEventSetGetContextData_v1_t
Parameters for retrieving raw context data associated with the most recent event.

nvmlEventSetGetContextInfo_v1_t
Parameters for retrieving context metadata associated with the most recent event.

nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t
Parameters for retrieving decoded GPU legacy-Xid context data.

nvmlEventSetWait_v3_t
Parameters for waiting on an event set with *nvmlEventSetWait_v3* .

nvmlExcludedDeviceInfo_t
Excluded GPU device information.

nvmlFBCSessionInfo_t
Structure to hold FBC session data.

nvmlFBCStats_t
Structure to hold frame buffer capture sessions stats.

nvmlFanSpeedInfo_v1_t
Fan speed info.

nvmlFieldValue_t
Information for a Field Value Sample.

nvmlGetCPER_v1_t
Input/output structure for *nvmlSystemGetCPER_v1* .

nvmlGetMemoryLimits_v1_t
Describes the current memory limits that are set for the device.

nvmlGpmMetric_t
GPM metric information.

nvmlGpmMetricsGet_t
GPM buffer information.

nvmlGpmSupport_t
GPM device information.

nvmlGpuDynamicPstatesInfo_t

nvmlGpuFabricClique_v1_t
Fabric clique entry.

nvmlGpuFabricInfo_t
Contains the device fabric information.

nvmlGpuFabricInfo_v2_t
GPU Fabric information (v2).

nvmlGpuFabricInfo_v3_t
GPU Fabric information (v3).

nvmlGpuFabricInfo_v4_t
GPU Fabric information (v4).

nvmlGpuInstanceInfo_t

nvmlGpuInstancePlacement_t

nvmlGpuInstanceProfileInfo_t
GPU instance profile information.

nvmlGpuInstanceProfileInfo_v2_t
GPU instance profile information (v2).

nvmlGpuInstanceProfileInfo_v3_t
GPU instance profile information (v3).

nvmlGpuOperationalEventConfig_v1_t
Configuration for registering structured GPU Operational Events with *nvmlEventSetRegisterGpuOperationalEvents_v1* .

nvmlGpuThermalSettings_t
Struct to hold the thermal sensor settings.

nvmlGpuUptime_v1_t
GPU Uptime.

nvmlGridLicensableFeature_t
Structure containing vGPU software licensable feature information.

nvmlGridLicensableFeatures_t
Structure to store vGPU software licensable features.

nvmlGridLicenseExpiry_t
Structure to store license expiry date and time values.

nvmlHostname_v1_t
Stores the hostname string assigned to a GPU device.

nvmlHwbcEntry_t
Description of HWBC entry.

nvmlLedState_t
LED states for an S-class unit.

nvmlMarginTemperature_v1_t
Margin temperature values.

nvmlMask255_t

nvmlMemory_t
Memory allocation information for a device (v1).

nvmlMemory_v2_t
Memory allocation information for a device (v2).

nvmlNvLinkInfo_v1_t
Struct to represent per device NVLINK information v1.

nvmlNvLinkInfo_v2_t
Struct to represent per device NVLINK information v2.

nvmlNvLinkInfo_v3_t
Struct to represent per device NVLINK information v3.

nvmlNvLinkPowerThres_t

nvmlNvLinkUtilizationControl_t
Struct to define the NVLINK counter controls.

nvmlNvlinkFirmwareInfo_t
Struct to represent NVLINK firmware information.

nvmlNvlinkFirmwareVersion_t
Struct to represent NVLINK firmware Semantic versioning and ucode type.

nvmlNvlinkGetBwMode_v1_t

nvmlNvlinkGetRemapTableInfo_v1_t
Structure for NVLink remap table information.

nvmlNvlinkLockRemapTable_v1_t
Structure for the NVLink remap table and MSE lock operation.

nvmlNvlinkNvlePrivateInfo_v1_t
Structure for NVLink NVLE private (opaque) information.

nvmlNvlinkSetBwModeAsync_v1_t
Describes the parameters involved to setting Nvlink RBM mode asynchronously.

nvmlNvlinkSetBwMode_v1_t

nvmlNvlinkSetNvleReady_v1_t
Structure for the NVLink NVLE ready state.

nvmlNvlinkSetupNvleEncryptionKey_v1_t
Structure for NVLink NVLE encryption key setup.

nvmlNvlinkSupportedBwModes_v1_t

nvmlNvlinkTelemetrySample_v1_t
Struct representing one (link, metric) telemetry request / response slot.

nvmlNvlinkTelemetrySamples_v1_t
Batched NVLink telemetry request.

nvmlObservedMetrics_t
Observed metrics from the power model, including initial DRAMCLK estimates and current measurements.

nvmlPRMCounterInput_v1_t
PRM counter input values.

nvmlPRMCounterList_v1_t
PRM Counter List Structure v1.

nvmlPRMCounterValue_v1_t
PRM Counter Value Structure.

nvmlPRMCounter_v1_t
PRM Counter Structure v1.

nvmlPRMTLV_v1_t
Main PRM input structure.

nvmlPSUInfo_t
Power usage information for an S-class unit.

nvmlPciInfoExt_v1_t
PCI information about a GPU device.

nvmlPciInfo_t
PCI information about a GPU device.

nvmlPdi_v1_t
Struct to represent the NVML PDI information.

nvmlPerfMetricControllerSample_t
Performance metrics sample from a controller, which can be either DLPPC 2x or PFPP 1x.

nvmlPerfMetricsDlppc2xSample_t
Performance metrics sample for DLPPC 2x controller.

nvmlPerfMetricsPfpp1xSample_t
Performance metrics sample for PFPP 1x controller.

nvmlPerfMetricsSample_t
Single performance metrics sample containing data from one or more controllers.

nvmlPerfMetricsSamples_v1_t
Collection of performance metrics samples (version 1).

nvmlPlatformInfo_v1_t
Structure to store platform information.

nvmlPlatformInfo_v2_t
Structure to store platform information (v2)

nvmlPmgrPwrTuple_t
Power tuple containing power consumption in milliwatts.

nvmlPowerSmoothingProfile_v1_t
Power Smoothing Structure for Profile information.

nvmlPowerSmoothingState_v1_t
Power Smoothing Structure for Feature Enablement.

nvmlPowerValue_v2_t
Contains the power management limit.

nvmlProcessDetailList_v1_t
Information about all running processes on the GPU for the given mode.

nvmlProcessDetail_v1_t
Information about running process on the GPU with protected memory.

nvmlProcessInfo_t
Information about running compute processes on the GPU.

nvmlProcessInfo_v1_t
Information about running compute processes on the GPU, legacy version for older versions of the API.

nvmlProcessUtilizationInfo_v1_t
Structure to store utilization value and process Id version 1.

nvmlProcessUtilizationSample_t
Structure to store utilization value and process Id.

nvmlProcessesUtilizationInfo_v1_t
Structure to store utilization and process ID for each running process version 1.

nvmlPwrModelMetricsDlppm1xDramclkEstimates_t
DRAMCLK estimates containing multiple estimated metrics for different DRAMCLK frequencies.

nvmlPwrModelMetricsDlppm1xPerf_t
Performance metrics for DLPPM 1x power model.

nvmlPwrModelMetricsDlppm1x_t
Complete power model metrics for DLPPM 1x, including rail metrics and TGP power.

nvmlPwrModelMetricsPfpp1x_t
Complete power model metrics for PFPP 1x, including estimated metrics and key operating points.

nvmlPwrModelMetricsSamplePfpp1x_t
Power model metrics sample for PFPP 1x, containing frequency inputs and estimated TGP.

nvmlPwrModelOperatingPointPfpp1x_t
Operating point for PFPP 1x power model, defining a frequency-power pair.

nvmlRailMetrics_t
Metrics for a single power rail, including frequency and utilization.

nvmlRemappedRowsInfo_v2_t

nvmlRepairStatus_v1_t
Struct to represent the NVML repair status.

nvmlRowRemapperHistogramValues_t
Possible values that classify the remap availability for each bank.

nvmlRusdSettings_v1_t
Stores RUSD polling configuration for a GPU.

nvmlSample_t
Information for Sample.

nvmlSetMemoryLimits_v1_t
Describes the memory limits that can be set for the device.

nvmlSystemConfComputeSettings_v1_t
Confidential Compute System settings.

nvmlSystemConfComputeSettings_v2_t
Confidential Compute System settings, version 2.

nvmlSystemDriverBranchInfo_v1_t
Structure to store Driver branch information.

nvmlSystemEventData_v1_t
nvmlSystemEventData_v1_t

nvmlSystemEventSetCreateRequest_v1_t
nvmlSystemEventSetCreateRequest

nvmlSystemEventSetFreeRequest_v1_t
nvmlSystemEventSetFreeRequest

nvmlSystemEventSetWaitRequest_v1_t
nvmlSystemEventSetWait

nvmlSystemRegisterEventRequest_v1_t
nvmlSystemRegisterEventRequest

nvmlTemperature_v1_t
Structure used to encapsulate temperature info.

nvmlUUID_v1_t
Struct to represent NVML UUID information.

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

nvmlUnitFanSpeeds_t
Fan speed readings for an entire S-class unit.

nvmlUnitInfo_t
Static S-class unit info.

nvmlUnrepairableMemoryStatus_v1_t
Struct to represent the NVML unrepairable memory status.

nvmlUtilization_t
Utilization information for a device.

nvmlVgpuCreatablePlacementInfo_v1_t
Structure to store creatable vGPU placement information version 1.

nvmlVgpuHeterogeneousMode_v1_t
Structure to store the vGPU heterogeneous mode of device version 1.

nvmlVgpuInstanceUtilizationInfo_v1_t
Structure to store Utilization Value and vgpuInstance Info Version 1.

nvmlVgpuInstanceUtilizationSample_t
Structure to store Utilization Value and vgpuInstance.

nvmlVgpuInstancesUtilizationInfo_v1_t
Structure to store recent utilization for vGPU instances running on a device version 1.

nvmlVgpuLicenseExpiry_t
Structure to store the vGPU license expiry details.

nvmlVgpuLicenseInfo_t

nvmlVgpuMetadata_t
vGPU metadata structure.

nvmlVgpuPgpuCompatibility_t
vGPU-pGPU compatibility structure

nvmlVgpuPgpuMetadata_t
Physical GPU metadata structure.

nvmlVgpuPlacementId_v1_t
Structure to store the placement ID of vGPU instance version 1.

nvmlVgpuPlacementList_v1_t
Structure to store the list of vGPU placements version 1.

nvmlVgpuPlacementList_v2_t
Structure to store the list of vGPU placements version 2.

nvmlVgpuProcessUtilizationInfo_v1_t
Structure to store Utilization Value, vgpuInstance and subprocess information for process running on vGPU instance version 1.

nvmlVgpuProcessUtilizationSample_t
Structure to store Utilization Value, vgpuInstance and subprocess information.

nvmlVgpuProcessesUtilizationInfo_v1_t
Structure to store recent utilization, vgpuInstance and subprocess information for processes running on vGPU instances active on a device version 1.

nvmlVgpuRuntimeState_v1_t
Structure to store the information of vGPU runtime state version 1.

nvmlVgpuSchedulerCapabilities_t
Structure to store the vGPU scheduler capabilities.

nvmlVgpuSchedulerGetState_t
Structure to store the vGPU scheduler state.

nvmlVgpuSchedulerLogEntry_t
Structure to store the state and logs of a software runlist.

nvmlVgpuSchedulerLogEntry_v2_t
Structure to store the state and logs of a software runlist.

nvmlVgpuSchedulerLogInfo_v1_t
Structure to store vGPU scheduler log information Version 1.

nvmlVgpuSchedulerLogInfo_v2_t
Structure to store vGPU scheduler log information.

nvmlVgpuSchedulerLog_t
Structure to store a vGPU software scheduler log.

nvmlVgpuSchedulerSetState_t
Structure to set the vGPU scheduler state.

nvmlVgpuSchedulerStateInfo_v1_t
Structure to store vGPU scheduler state information Version 1.

nvmlVgpuSchedulerStateInfo_v2_t
Structure to store vGPU scheduler state information.

nvmlVgpuSchedulerState_v1_t
Structure to set vGPU scheduler state information version 1.

nvmlVgpuSchedulerState_v2_t
Structure to set vGPU scheduler state information.

nvmlVgpuTypeBar1Info_v1_t
Structure to store BAR1 size information of vGPU type Version 1.

nvmlVgpuTypeIdInfo_v1_t
Structure to store the vGPU type IDs version 1.

nvmlVgpuTypeMaxInstance_v1_t
Structure to store the maximum number of possible vGPU type IDs version 1.

nvmlVgpuVersion_t
Structure representing range of vGPU versions.

nvmlViolationTime_t
Struct to hold perf policy violation status data.

nvmlWorkloadPowerProfileCurrentProfiles_v1_t
Current Profiles.

nvmlWorkloadPowerProfileInfo_v1_t
Profile Metadata.

nvmlWorkloadPowerProfileProfilesInfo_v1_t
Profiles Info.

nvmlWorkloadPowerProfileRequestedProfiles_v1_t
Requested Profiles.

nvmlWorkloadPowerProfileUpdateProfiles_v1_t
Update Profiles.

5.2.4. Unions

5.2.4.1 nvmlUUIDValue_t

union **nvmlUUIDValue_t**

#include </home/rachnap/pdf-build/docs-gpu-deploy/docs/nvml-api/headers/nvml.h> Union to represent different UUID values.

Public Members

char **str**[41]
ASCII format value.

unsigned char **bytes**[16]
Binary format value.

5.2.4.2 nvmlValue_t

union **nvmlValue_t**

#include </home/rachnap/pdf-build/docs-gpu-deploy/docs/nvml-api/headers/nvml.h> Union to represent different types of Value.

Public Members

double **dVal**
If the value is double.

int **siVal**
If the value is signed int.

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.

signed long long **sllVal**

If the value is signed long long.

unsigned short **usVal**

If the value is unsigned short.

5.2.4.3 nvmlVgpuSchedulerParams_t

union **nvmlVgpuSchedulerParams_t**

#include </home/rachnap/pdf-build/docs-gpu-deploy/docs/nvml-api/headers/nvml.h> Union to represent the vGPU Scheduler Parameters.

Public Members

unsigned int **avgFactor**

Average factor in compensating the timeslice for Adaptive Round Robin mode. 0 when there is no active scheduling.

unsigned int **timeslice**

The timeslice in ns for each software run list as configured, or the default value otherwise. 0 when there is no active scheduling.

struct *nvmlVgpuSchedulerParams_t*::[anonymous] **vgpuSchedDataWithARR**

struct *nvmlVgpuSchedulerParams_t*::[anonymous] **vgpuSchedData**

5.2.4.4 nvmlVgpuSchedulerSetParams_t

union **nvmlVgpuSchedulerSetParams_t**

#include </home/rachnap/pdf-build/docs-gpu-deploy/docs/nvml-api/headers/nvml.h> Union to represent the vGPU Scheduler set Parameters.

Public Members

unsigned int **avgFactor**

Average factor in compensating the timeslice for Adaptive Round Robin mode. 0 or unspecified uses default.

unsigned int **frequency**

Frequency for Adaptive Round Robin mode. 0 or unspecified uses default.

struct *nvmlVgpuSchedulerSetParams_t*::[anonymous] **vgpuSchedDataWithARR**

unsigned int **timeslice**

The timeslice in ns(Nanoseconds) for each software run list as configured, or the default value if unspecified or set to 0.

struct *nvmlVgpuSchedulerSetParams_t*::[anonymous] **vgpuSchedData**

nvmlUUIDValue_t

Union to represent different UUID values.

nvmlValue_t

Union to represent different types of Value.

nvmlVgpuSchedulerParams_t

Union to represent the vGPU Scheduler Parameters.

nvmlVgpuSchedulerSetParams_t

Union to represent the vGPU Scheduler set Parameters.

Chapter 6. Notices

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND 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. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

6.1. Trademarks

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

Copyright

©2007-2026, NVIDIA Corporation & affiliates. All rights reserved

Index

I

INVALID_GPU_INSTANCE_ID (*C macro*), 436
INVALID_GPU_INSTANCE_PROFILE_ID (*C macro*), 436

M

MAX_CLK_DOMAINS (*C macro*), 77

N

NVML_ADAPTIVE_CLOCKING_INFO_STATUS_DISABLED (*C macro*), 77
NVML_ADAPTIVE_CLOCKING_INFO_STATUS_ENABLED (*C macro*), 77
NVML_AFFINITY_SCOPE_NODE (*C macro*), 96
NVML_AFFINITY_SCOPE_SOCKET (*C macro*), 96
NVML_BUS_TYPE_AGP (*C macro*), 77
NVML_BUS_TYPE_FPCI (*C macro*), 77
NVML_BUS_TYPE_PCI (*C macro*), 77
NVML_BUS_TYPE_PCIE (*C macro*), 77
NVML_BUS_TYPE_UNKNOWN (*C macro*), 78
NVML_C2C_POWER_STATE_FULL_POWER (*C macro*), 285
NVML_C2C_POWER_STATE_LOW_POWER (*C macro*), 285
NVML_CC_ACCEPTING_CLIENT_REQUESTS_FALSE (*C macro*), 49
NVML_CC_ACCEPTING_CLIENT_REQUESTS_TRUE (*C macro*), 49
NVML_CC_CEC_ATTESTATION_REPORT_NOT_PRESENT (*C macro*), 49
NVML_CC_CEC_ATTESTATION_REPORT_PRESENT (*C macro*), 49
NVML_CC_GPU_ATTESTATION_REPORT_SIZE (*C macro*), 49
NVML_CC_GPU_CEC_ATTESTATION_REPORT_SIZE (*C macro*), 49
NVML_CC_GPU_CEC_NONCE_SIZE (*C macro*), 49
NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MAX (*C macro*), 49
NVML_CC_KEY_ROTATION_THRESHOLD_ATTACKER_ADVANTAGE_MIN (*C macro*), 49
NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV (*C macro*), 50
NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV_SNP (*C macro*), 50
NVML_CC_SYSTEM_CPU_CAPS_AMD_SEV_TIO (*C macro*), 50
NVML_CC_SYSTEM_CPU_CAPS_AMD_SNP_VTOM (*C macro*), 50
NVML_CC_SYSTEM_CPU_CAPS_ARM_CCA (*C macro*), 50
NVML_CC_SYSTEM_CPU_CAPS_INTEL_TDX (*C macro*), 50
NVML_CC_SYSTEM_CPU_CAPS_INTEL_TDX_CONNECT (*C macro*), 50
NVML_CC_SYSTEM_CPU_CAPS_NONE (*C macro*), 50
NVML_CC_SYSTEM_DEVTOOLS_MODE_OFF (*C macro*), 50
NVML_CC_SYSTEM_DEVTOOLS_MODE_ON (*C macro*), 50
NVML_CC_SYSTEM_ENVIRONMENT_PROD (*C macro*), 50
NVML_CC_SYSTEM_ENVIRONMENT_SIM (*C macro*), 50
NVML_CC_SYSTEM_ENVIRONMENT_UNAVAILABLE (*C macro*), 50
NVML_CC_SYSTEM_FEATURE_BMSAI_ENABLED (*C macro*), 50
NVML_CC_SYSTEM_FEATURE_DISABLED (*C macro*), 51
NVML_CC_SYSTEM_FEATURE_ENABLED (*C macro*), 51
NVML_CC_SYSTEM_GPUS_CC_BMSAI_CAPABLE (*C macro*), 51
NVML_CC_SYSTEM_GPUS_CC_CAPABLE (*C macro*), 51
NVML_CC_SYSTEM_GPUS_CC_NOT_CAPABLE (*C macro*), 51
NVML_CC_SYSTEM_MULTIGPU_NONE (*C macro*), 51
NVML_CC_SYSTEM_MULTIGPU_NVLE (*C macro*), 51
NVML_CC_SYSTEM_MULTIGPU_PROTECTED_PCIE (*C macro*), 51
NVML_CC_SYSTEM_TDISP_STATE_DISABLED (*C macro*), 51
NVML_CC_SYSTEM_TDISP_STATE_NOT_READY (*C macro*), 51

NVML_CC_SYSTEM_TDISP_STATE_READY (*C macro*), 51
NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_COUNT (*C macro*), 319
NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_SHARED (*C macro*), 319
NVML_COMPUTE_INSTANCE_ID_ANY (*C macro*), 244
NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE_REV1 (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_2_SLICE (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_3_SLICE (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_4_SLICE (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_6_SLICE (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_7_SLICE (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_7_SLICE_NVL (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_8_SLICE (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_CAPS_GFX (*C macro*), 319
NVML_COMPUTE_INSTANCE_PROFILE_COUNT (*C macro*), 320
NVML_CPER_CURSOR_HANDLE_INIT (*C macro*), 46
NVML_CUDA_DRIVER_VERSION_MAJOR (*C macro*), 415
NVML_CUDA_DRIVER_VERSION_MINOR (*C macro*), 415
NVML_DEVICE_ARCH_ADA (*C macro*), 78
NVML_DEVICE_ARCH_AMPERE (*C macro*), 78
NVML_DEVICE_ARCH_BLACKWELL (*C macro*), 78
NVML_DEVICE_ARCH_DLA (*C macro*), 78
NVML_DEVICE_ARCH_DLA2 (*C macro*), 78
NVML_DEVICE_ARCH_HOPPER (*C macro*), 78
NVML_DEVICE_ARCH_KEPLER (*C macro*), 78
NVML_DEVICE_ARCH_MAXWELL (*C macro*), 78
NVML_DEVICE_ARCH_NPU3 (*C macro*), 78
NVML_DEVICE_ARCH_PASCAL (*C macro*), 78
NVML_DEVICE_ARCH_RUBIN (*C macro*), 78
NVML_DEVICE_ARCH_TURING (*C macro*), 78
NVML_DEVICE_ARCH_UNKNOWN (*C macro*), 78
NVML_DEVICE_ARCH_VOLTA (*C macro*), 79
NVML_DEVICE_HOSTNAME_BUFFER_SIZE (*C macro*), 79
NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE (*C macro*), 53
NVML_DEVICE_MEMORY_LIMIT_MAX (*C macro*), 221
NVML_DEVICE_MIG_DISABLE (*C macro*), 320
NVML_DEVICE_MIG_ENABLE (*C macro*), 320
NVML_DEVICE_NAME_BUFFER_SIZE (*C macro*), 53
NVML_DEVICE_NAME_V2_BUFFER_SIZE (*C macro*), 53
NVML_DEVICE_PART_NUMBER_BUFFER_SIZE (*C macro*), 53
NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE (*C macro*), 221
NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE (*C macro*), 221
NVML_DEVICE_PCI_BUS_ID_FMT (*C macro*), 221
NVML_DEVICE_PCI_BUS_ID_FMT_ARGS (*C macro*), 221
NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT (*C macro*), 221
NVML_DEVICE_SERIAL_BUFFER_SIZE (*C macro*), 53
NVML_DEVICE_UUID_ASCII_LEN (*C macro*), 221
NVML_DEVICE_UUID_BINARY_LEN (*C macro*), 222
NVML_DEVICE_UUID_BUFFER_SIZE (*C macro*), 53
NVML_DEVICE_UUID_V2_BUFFER_SIZE (*C macro*), 53
NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE (*C macro*), 53
NVML_DOUBLE_BIT_ECC (*C macro*), 79
NVML_FAN_POLICY_MANUAL (*C macro*), 79
NVML_FAN_POLICY_TEMPERATURE_CONTINOUS_SW (*C macro*), 79
NVML_FI_DEV_ACTIVE_BANK_REMAPPINGS (*C macro*), 285
NVML_FI_DEV_BANK_REMAPPER_HISTOGRAM_MAX (*C macro*), 285
NVML_FI_DEV_BANK_REMAPPER_HISTOGRAM_NONE (*C macro*), 285
NVML_FI_DEV_C2C_LINK_COUNT (*C macro*), 285
NVML_FI_DEV_C2C_LINK_ERROR_INTR (*C macro*), 285
NVML_FI_DEV_C2C_LINK_ERROR_REPLAY (*C macro*), 285
NVML_FI_DEV_C2C_LINK_ERROR_REPLAY_B2B (*C macro*), 285
NVML_FI_DEV_C2C_LINK_GET_MAX_BW (*C macro*), 285
NVML_FI_DEV_C2C_LINK_GET_STATUS (*C macro*), 285
NVML_FI_DEV_C2C_LINK_POWER_STATE (*C macro*), 285
NVML_FI_DEV_CLOCKS_EVENT_REASON_HW_POWER_BRAKE_SLOW (*C macro*), 285
NVML_FI_DEV_CLOCKS_EVENT_REASON_HW_THERM_SLOWDOWN (*C macro*), 285
NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_POWER_CAP

(*C macro*), 285
NVML_FI_DEV_CLOCKS_EVENT_REASON_SW_THERM_SLOWDOWN (*C macro*), 286
NVML_FI_DEV_CLOCKS_EVENT_REASON_SYNC_BOOST (*C macro*), 286
NVML_FI_DEV_DRAIN_AND_RESET_STATUS (*C macro*), 286
NVML_FI_DEV_ECC_CURRENT (*C macro*), 286
NVML_FI_DEV_ECC_DBE_AGG_CBU (*C macro*), 286
NVML_FI_DEV_ECC_DBE_AGG_DEV (*C macro*), 286
NVML_FI_DEV_ECC_DBE_AGG_L1 (*C macro*), 286
NVML_FI_DEV_ECC_DBE_AGG_L2 (*C macro*), 286
NVML_FI_DEV_ECC_DBE_AGG_REG (*C macro*), 286
NVML_FI_DEV_ECC_DBE_AGG_TEX (*C macro*), 286
NVML_FI_DEV_ECC_DBE_AGG_TOTAL (*C macro*), 286
NVML_FI_DEV_ECC_DBE_VOL_CBU (*C macro*), 286
NVML_FI_DEV_ECC_DBE_VOL_DEV (*C macro*), 286
NVML_FI_DEV_ECC_DBE_VOL_L1 (*C macro*), 286
NVML_FI_DEV_ECC_DBE_VOL_L2 (*C macro*), 287
NVML_FI_DEV_ECC_DBE_VOL_REG (*C macro*), 287
NVML_FI_DEV_ECC_DBE_VOL_TEX (*C macro*), 287
NVML_FI_DEV_ECC_DBE_VOL_TOTAL (*C macro*), 287
NVML_FI_DEV_ECC_PENDING (*C macro*), 287
NVML_FI_DEV_ECC_SBE_AGG_DEV (*C macro*), 287
NVML_FI_DEV_ECC_SBE_AGG_L1 (*C macro*), 287
NVML_FI_DEV_ECC_SBE_AGG_L2 (*C macro*), 287
NVML_FI_DEV_ECC_SBE_AGG_REG (*C macro*), 287
NVML_FI_DEV_ECC_SBE_AGG_TEX (*C macro*), 287
NVML_FI_DEV_ECC_SBE_AGG_TOTAL (*C macro*), 287
NVML_FI_DEV_ECC_SBE_VOL_DEV (*C macro*), 287
NVML_FI_DEV_ECC_SBE_VOL_L1 (*C macro*), 287
NVML_FI_DEV_ECC_SBE_VOL_L2 (*C macro*), 287
NVML_FI_DEV_ECC_SBE_VOL_REG (*C macro*), 287
NVML_FI_DEV_ECC_SBE_VOL_TEX (*C macro*), 288
NVML_FI_DEV_ECC_SBE_VOL_TOTAL (*C macro*), 288
NVML_FI_DEV_EDPP_MULTIPLIER (*C macro*), 288
NVML_FI_DEV_ENERGY (*C macro*), 288
NVML_FI_DEV_GET_GPU_RECOVERY_ACTION (*C macro*), 288
NVML_FI_DEV_GPU_TEMP (*C macro*), 288
NVML_FI_DEV_INACTIVE_BANK_REMAPPINGS (*C macro*), 288
NVML_FI_DEV_IS_MIG_MODE_INDEPENDENT_MIG_QUERY_CAPABLE (*C macro*), 288
NVML_FI_DEV_IS_RESETLESS_MIG_SUPPORTED (*C macro*), 288
NVML_FI_DEV_MCLK_MIN_SWITCH_INTERVAL_MILLISECONDS (*C macro*), 288
NVML_FI_DEV_MCLK_SWITCH_TYPE (*C macro*), 288
NVML_FI_DEV_MEMORY_TEMP (*C macro*), 288
NVML_FI_DEV_MMA_STALL_PERCENT (*C macro*), 288
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L0 (*C macro*), 288
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L1 (*C macro*), 288
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L2 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L3 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L4 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L5 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L6 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L7 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L8 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L9 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L10 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L11 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_TOTAL (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L0 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L1 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L2 (*C macro*), 290
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L3 (*C macro*), 290
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L4 (*C macro*), 290
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L5 (*C macro*), 290
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L6 (*C macro*), 290
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L7 (*C macro*), 290
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L8 (*C macro*), 290
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L9 (*C macro*), 290
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L10 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L11 (*C macro*), 289
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_TOTAL

(*C macro*), 290
NVML_FI_DEV_NVLINK_COUNT_BUFFER_OVERRUN_ERRORS (*C macro*), 290
NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_BER (*C macro*), 290
NVML_FI_DEV_NVLINK_COUNT_EFFECTIVE_ERRORS (*C macro*), 290
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_0 (*C macro*), 290
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_1 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_2 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_3 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_4 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_5 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_6 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_7 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_8 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_9 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_10 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_11 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_12 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_13 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_14 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_FEC_HISTORY_15 (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_EVENTS (*C macro*), 291
NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_FAILED_EVENTS (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_LINK_RECOVERY_SUCCESSFUL_EVENTS (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_LOCAL_LINK_INTEGRITY_ERRORS (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_MALFORMED_PACKET_ERRORS (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RAW_BER (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE0 (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE0_V2 (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE1 (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RAW_BER_LANE1_V2 (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RAW_BER_V2 (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RAW_ERRORS_LANE0 (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RAW_ERRORS_LANE1 (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RCV_BYTES (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RCV_ERRORS (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RCV_GENERAL_ERRORS (*C macro*), 292
NVML_FI_DEV_NVLINK_COUNT_RCV_PACKETS (*C macro*), 293
NVML_FI_DEV_NVLINK_COUNT_RCV_REMOTE_ERRORS (*C macro*), 293
NVML_FI_DEV_NVLINK_COUNT_SYMBOL_BER (*C macro*), 293
NVML_FI_DEV_NVLINK_COUNT_SYMBOL_ERRORS (*C macro*), 293
NVML_FI_DEV_NVLINK_COUNT_VL15_DROPPED (*C macro*), 293
NVML_FI_DEV_NVLINK_COUNT_XMIT_BYTES (*C macro*), 293
NVML_FI_DEV_NVLINK_COUNT_XMIT_DISCARDS (*C macro*), 293
NVML_FI_DEV_NVLINK_COUNT_XMIT_PACKETS (*C macro*), 293
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L0 (*C macro*), 293
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L1 (*C macro*), 293
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L2 (*C macro*), 293
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L3 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L4 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L5 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L6 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L7 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L8 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L9 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L10

(*C macro*), 293
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L1 (*C macro*), 293
NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_TOTAL (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L0 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L1 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L2 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L3 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L4 (*C macro*), 295
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L5 (*C macro*), 295
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L6 (*C macro*), 295
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L7 (*C macro*), 295
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L8 (*C macro*), 295
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L9 (*C macro*), 294
NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_TOTAL (*C macro*), 295
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L0 (*C macro*), 295
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L1 (*C macro*), 295
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L2 (*C macro*), 295
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L3 (*C macro*), 296
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L4 (*C macro*), 296
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L5 (*C macro*), 296
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L6 (*C macro*), 296
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L7 (*C macro*), 296
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L8 (*C macro*), 295
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_L9 (*C macro*), 295
NVML_FI_DEV_NVLINK_ECC_DATA_ERROR_COUNT_TOTAL (*C macro*), 296
NVML_FI_DEV_NVLINK_ERROR_DL_CRC (*C macro*), 296
NVML_FI_DEV_NVLINK_ERROR_DL_RECOVERY (*C macro*), 296
NVML_FI_DEV_NVLINK_ERROR_DL_REPLAY (*C macro*), 296
NVML_FI_DEV_NVLINK_GET_DATA_RATE (*C macro*), 296
NVML_FI_DEV_NVLINK_GET_POWER_STATE (*C macro*), 296
NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD (*C macro*), 296
NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_MAX (*C macro*), 297
NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_MIN (*C macro*), 297
NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_SUPPORTED (*C macro*), 297
NVML_FI_DEV_NVLINK_GET_POWER_THRESHOLD_UNITS (*C macro*), 297
NVML_FI_DEV_NVLINK_GET_SPEED (*C macro*), 297
NVML_FI_DEV_NVLINK_GET_STATE (*C macro*), 297
NVML_FI_DEV_NVLINK_GET_VERSION (*C macro*), 297
NVML_FI_DEV_NVLINK_LINK_COUNT (*C macro*), 297
NVML_FI_DEV_NVLINK_PLR_XMIT_BLOCKS (*C macro*), 297
NVML_FI_DEV_NVLINK_PLR_XMIT_RETRY_BLOCKS (*C macro*), 297
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L0 (*C macro*), 297
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L1 (*C macro*), 297
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L2 (*C macro*), 298
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L3 (*C macro*), 298
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L4 (*C macro*), 298
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L5 (*C macro*), 298
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L6 (*C macro*), 298
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L7 (*C macro*), 298
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L8 (*C macro*), 298
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L9

(*C macro*), 298
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L7 (*C macro*), 297
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L8 (*C macro*), 297
NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_TOTAL (*C macro*), 298
NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID (*C macro*), 298
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L0 (*C macro*), 298
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L1 (*C macro*), 298
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L2 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L3 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L4 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L5 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L6 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L7 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L8 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L9 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L10 (*C macro*), 298
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L11 (*C macro*), 299
NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_TOTAL (*C macro*), 299
NVML_FI_DEV_NVLINK_SPEED_MBPS_COMMON (*C macro*), 299
NVML_FI_DEV_NVLINK_SPEED_MBPS_L0 (*C macro*), 299
NVML_FI_DEV_NVLINK_SPEED_MBPS_L1 (*C macro*), 299
NVML_FI_DEV_NVLINK_SPEED_MBPS_L2 (*C macro*), 300
NVML_FI_DEV_NVLINK_SPEED_MBPS_L3 (*C macro*), 300
NVML_FI_DEV_NVLINK_SPEED_MBPS_L4 (*C macro*), 300
NVML_FI_DEV_NVLINK_SPEED_MBPS_L5 (*C macro*), 300
NVML_FI_DEV_NVLINK_SPEED_MBPS_L6 (*C macro*), 300
NVML_FI_DEV_NVLINK_SPEED_MBPS_L7 (*C macro*), 300
NVML_FI_DEV_NVLINK_SPEED_MBPS_L8 (*C macro*), 300
NVML_FI_DEV_NVLINK_SPEED_MBPS_L9 (*C macro*), 300
NVML_FI_DEV_NVLINK_SPEED_MBPS_L10 (*C macro*), 299
NVML_FI_DEV_NVLINK_SPEED_MBPS_L11 (*C macro*), 300
NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_RX (*C macro*), 300
NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_TX (*C macro*), 300
NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_RX (*C macro*), 300
NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_TX (*C macro*), 300
NVML_FI_DEV_NVSWITCH_CONNECTED_LINK_COUNT (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_BAD_DLLP (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_BAD_TLP (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_CORRECTABLE_ERRORS (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_FATAL_ERROR (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_LANE_ERROR (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_LCRC_ERROR (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_NAKS_RECEIVED (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_NAKS_SENT (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_NON_FATAL_ERROR (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_RECEIVER_ERROR (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_RX_BYTES (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_TX_BYTES (*C macro*), 301
NVML_FI_DEV_PCIE_COUNT_UNSUPPORTED_REQ (*C macro*), 301
NVML_FI_DEV_PCIE_INBOUND_ATOMICS_MASK (*C macro*), 301
NVML_FI_DEV_PCIE_L0_TO_RECOVERY_COUNTER (*C macro*), 302
NVML_FI_DEV_PCIE_OUTBOUND_ATOMICS_MASK (*C macro*), 302
NVML_FI_DEV_PCIE_REPLAY_COUNTER (*C macro*), 302
NVML_FI_DEV_PCIE_REPLAY_ROLLOVER_COUNTER (*C macro*), 302
NVML_FI_DEV_PENDING_BANK_REMAPPING (*C*

macro), 302
NVML_FI_DEV_PERF_POLICY_BOARD_LIMIT (*C macro*), 302
NVML_FI_DEV_PERF_POLICY_LOW_UTILIZATION (*C macro*), 302
NVML_FI_DEV_PERF_POLICY_POWER (*C macro*), 302
NVML_FI_DEV_PERF_POLICY_RELIABILITY (*C macro*), 302
NVML_FI_DEV_PERF_POLICY_SYNC_BOOST (*C macro*), 302
NVML_FI_DEV_PERF_POLICY_THERMAL (*C macro*), 302
NVML_FI_DEV_PERF_POLICY_TOTAL_APP_CLOCKS (*C macro*), 302
NVML_FI_DEV_PERF_POLICY_TOTAL_BASE_CLOCKS (*C macro*), 302
NVML_FI_DEV_POWER_AVERAGE (*C macro*), 302
NVML_FI_DEV_POWER_CURRENT_LIMIT (*C macro*), 303
NVML_FI_DEV_POWER_DEFAULT_LIMIT (*C macro*), 303
NVML_FI_DEV_POWER_INSTANT (*C macro*), 303
NVML_FI_DEV_POWER_MAX_LIMIT (*C macro*), 303
NVML_FI_DEV_POWER_MIN_LIMIT (*C macro*), 303
NVML_FI_DEV_POWER_REQUESTED_LIMIT (*C macro*), 303
NVML_FI_DEV_POWER_SYNC_BALANCING_AF (*C macro*), 303
NVML_FI_DEV_POWER_SYNC_BALANCING_FREQ (*C macro*), 303
NVML_FI_DEV_REMAPPED_COR (*C macro*), 303
NVML_FI_DEV_REMAPPED_FAILURE (*C macro*), 303
NVML_FI_DEV_REMAPPED_PENDING (*C macro*), 303
NVML_FI_DEV_REMAPPED_ROWS_COR_INACTIVE (*C macro*), 303
NVML_FI_DEV_REMAPPED_ROWS_UNC_INACTIVE (*C macro*), 303
NVML_FI_DEV_REMAPPED_UNC (*C macro*), 303
NVML_FI_DEV_RESET_STATUS (*C macro*), 304
NVML_FI_DEV_RETIRED_DBE (*C macro*), 304
NVML_FI_DEV_RETIRED_PENDING (*C macro*), 304
NVML_FI_DEV_RETIRED_PENDING_DBE (*C macro*), 304
NVML_FI_DEV_RETIRED_PENDING_SBE (*C macro*), 304
NVML_FI_DEV_RETIRED_SBE (*C macro*), 304
NVML_FI_DEV_TEMPERATURE_GPU_MAX_TLIMIT (*C macro*), 304
NVML_FI_DEV_TEMPERATURE_MEM_MAX_TLIMIT (*C macro*), 304
NVML_FI_DEV_TEMPERATURE_SHUTDOWN_TLIMIT (*C macro*), 304
NVML_FI_DEV_TEMPERATURE_SLOWDOWN_TLIMIT (*C macro*), 304
NVML_FI_DEV_TOTAL_ENERGY_CONSUMPTION (*C macro*), 304
NVML_FI_MAX (*C macro*), 304
NVML_FI_PWR_SMOOTHING_ACTIVE_PRESET_PROFILE (*C macro*), 304
NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PERCENT_TMP_FL (*C macro*), 304
NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR (*C macro*), 305
NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR (*C macro*), 305
NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_PRIMARY_FLOOR (*C macro*), 305
NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_DOWN_HYST (*C macro*), 305
NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_DOWN_RATE (*C macro*), 305
NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_RAMP_UP_RATE (*C macro*), 305
NVML_FI_PWR_SMOOTHING_ADMIN_OVERRIDE_SECONDARY_POWE (*C macro*), 305
NVML_FI_PWR_SMOOTHING_APPLIED_TMP_CEIL (*C macro*), 305
NVML_FI_PWR_SMOOTHING_APPLIED_TMP_FLOOR (*C macro*), 305
NVML_FI_PWR_SMOOTHING_DELAYED_PWR_SMOOTHING_SUPPORT (*C macro*), 305
NVML_FI_PWR_SMOOTHING_ENABLED (*C macro*), 305
NVML_FI_PWR_SMOOTHING_HW_CIRCUITRY_PERCENT_LIFETIME (*C macro*), 306
NVML_FI_PWR_SMOOTHING_IMM_RAMP_DOWN_ENABLED (*C macro*), 306
NVML_FI_PWR_SMOOTHING_LIMITED_TIME (*C macro*), 306
NVML_FI_PWR_SMOOTHING_MAX_NUM_PRESET_PROFILES (*C macro*), 306
NVML_FI_PWR_SMOOTHING_MAX_PERCENT_TMP_FLOOR_SETTING (*C macro*), 306
NVML_FI_PWR_SMOOTHING_MIN_PERCENT_TMP_FLOOR_SETTING (*C macro*), 306
NVML_FI_PWR_SMOOTHING_MIN_PRIMARY_FLOOR_ACT_OFFSET (*C macro*), 306
NVML_FI_PWR_SMOOTHING_MIN_PRIMARY_FLOOR_ACT_POINT (*C macro*), 306
NVML_FI_PWR_SMOOTHING_PRIMARY_POWER_FLOOR (*C macro*), 306
NVML_FI_PWR_SMOOTHING_PRIV_LVL (*C macro*), 306
NVML_FI_PWR_SMOOTHING_PROFILE_PERCENT_TMP_FLOOR (*C macro*), 306

NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_LIFETIME (C macro), 306
NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_UTILIZATION (C macro), 307
NVML_FI_PWR_SMOOTHING_PROFILE_PRIMARY_FLOOR_WEIGHT (C macro), 307
NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_DOWN_HYST_VAL (C macro), 307
NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_DOWN_RATE (C macro), 307
NVML_FI_PWR_SMOOTHING_PROFILE_RAMP_UP_RATE (C macro), 307
NVML_FI_PWR_SMOOTHING_PROFILE_SECONDARY_FLOOR_UTILIZATION (C macro), 307
NVML_FI_PWR_SMOOTHING_SECONDARY_POWER_FLOOR (C macro), 307
NVML_FI_PWR_SMOOTHING_SOC_POWER_SMOOTHING_ENABLED (C macro), 307
NVML_FI_PWR_SMOOTHING_WINDOW_MULTIPLIER (C macro), 307
NVML_GPM_METRICS_GET_VERSION (C macro), 373
NVML_GPM_SUPPORT_VERSION (C macro), 373
NVML_GPU_ATTESTATION_CERT_CHAIN_SIZE (C macro), 51
NVML_GPU_CERT_CHAIN_SIZE (C macro), 51
NVML_GPU_FABRIC_CLIQUE_MAX (C macro), 264
NVML_GPU_FABRIC_CLIQUE_TYPE_MULTICAST_LOGICAL_ENDPOINT (C macro), 264
NVML_GPU_FABRIC_CLIQUE_TYPE_MULTICAST_POINTER (C macro), 264
NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_LOGICAL_ENDPOINT (C macro), 264
NVML_GPU_FABRIC_CLIQUE_TYPE_UNICAST_POINTER (C macro), 265
NVML_GPU_FABRIC_HEALTH_GET (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_ACCESS_TIMEOUT_RECOVERY (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_ACCESS_TIMEOUT_RECOVERY_NOT_SUPPORTED (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_FALSE (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_NOT_SUPPORTED (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_DEGRADED_BW_TRUE (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_CONNECTED (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_DISCONNECTED (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_GFM_STATE_NOT_SUPPORTED (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_NOT_SUPPORTED (C macro), 265
NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_TRUE (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_UNSUPPORTED (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_UNSUPPORTED (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_UNSUPPORTED (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_UNSUPPORTED (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_INCORRECT_CONFIGURATION_UNSUPPORTED (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_FALSE (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_NOT_SUPPORTED (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_PARTITION_ASSIGNED_TRUE (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_FALSE (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_NOT_SUPPORTED (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_RECOVERY_TRUE (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_FALSE (C macro), 266
NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_NOT_SUPPORTED (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_ROUTE_UNHEALTHY_TRUE (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ACCESS_TIMEOUT_RECOVERY (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_DEGRADED_BW (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_GFM_STATE (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_INCORRECT_CONFIGURATION (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_PARTITION_ASSIGNED (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ROUTE_RECOVERY (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_SHIFT_ROUTE_UNHEALTHY (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ACCESS_TIMEOUT_RECOVERY (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_DEGRADED_BW (C macro), 267

NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_GFM_STATE (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_INCORRECT_CONFIGURATION (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_PARTITION_ASSIGNED (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ROUTE_RECOVERY (C macro), 267
NVML_GPU_FABRIC_HEALTH_MASK_WIDTH_ROUTE_UNHEALTHY (C macro), 268
NVML_GPU_FABRIC_HEALTH_SUMMARY_HEALTHY (C macro), 268
NVML_GPU_FABRIC_HEALTH_SUMMARY_LIMITED_CAPACITY (C macro), 268
NVML_GPU_FABRIC_HEALTH_SUMMARY_NOT_SUPPORTED (C macro), 268
NVML_GPU_FABRIC_HEALTH_SUMMARY_UNHEALTHY (C macro), 268
NVML_GPU_FABRIC_HEALTH_TEST (C macro), 268
NVML_GPU_FABRIC_STATE_COMPLETED (C macro), 268
NVML_GPU_FABRIC_STATE_IN_PROGRESS (C macro), 268
NVML_GPU_FABRIC_STATE_NOT_STARTED (C macro), 268
NVML_GPU_FABRIC_STATE_NOT_SUPPORTED (C macro), 268
NVML_GPU_FABRIC_UUID_LEN (C macro), 268
NVML_GPU_INSTANCE_ID_ANY (C macro), 244
NVML_GPU_INSTANCE_PROFILE_1_SLICE (C macro), 320
NVML_GPU_INSTANCE_PROFILE_1_SLICE_ALL_ME (C macro), 320
NVML_GPU_INSTANCE_PROFILE_1_SLICE_GFX (C macro), 320
NVML_GPU_INSTANCE_PROFILE_1_SLICE_NO_ME (C macro), 320
NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV1 (C macro), 320
NVML_GPU_INSTANCE_PROFILE_1_SLICE_REV2 (C macro), 320
NVML_GPU_INSTANCE_PROFILE_2_SLICE (C macro), 320
NVML_GPU_INSTANCE_PROFILE_2_SLICE_ALL_ME (C macro), 320
NVML_GPU_INSTANCE_PROFILE_2_SLICE_GFX (C macro), 320
NVML_GPU_INSTANCE_PROFILE_2_SLICE_NO_ME (C macro), 320
NVML_GPU_INSTANCE_PROFILE_2_SLICE_REV1 (C macro), 321
NVML_GPU_INSTANCE_PROFILE_3_SLICE (C macro), 321
NVML_GPU_INSTANCE_PROFILE_3_SLICE_GFX (C macro), 321
NVML_GPU_INSTANCE_PROFILE_4_SLICE (C macro), 321
NVML_GPU_INSTANCE_PROFILE_4_SLICE_GFX (C macro), 321
NVML_GPU_INSTANCE_PROFILE_6_SLICE (C macro), 321
NVML_GPU_INSTANCE_PROFILE_7_SLICE (C macro), 321
NVML_GPU_INSTANCE_PROFILE_8_SLICE (C macro), 321
NVML_GPU_INSTANCE_PROFILE_CAPS_GFX (C macro), 321
NVML_GPU_INSTANCE_PROFILE_CAPS_P2P (C macro), 321
NVML_GPU_INSTANCE_PROFILE_COUNT (C macro), 321
NVML_GPU_INTSTANCE_PROFILE_CAPS_P2P (C macro), 321
NVML_GRID_LICENSE_BUFFER_SIZE (C macro), 436
NVML_GRID_LICENSE_EXPIRY_INVALID (C macro), 437
NVML_GRID_LICENSE_EXPIRY_NOT_APPLICABLE (C macro), 437
NVML_GRID_LICENSE_EXPIRY_NOT_AVAILABLE (C macro), 437
NVML_GRID_LICENSE_EXPIRY_PERMANENT (C macro), 437
NVML_GRID_LICENSE_EXPIRY_VALID (C macro), 438
NVML_GRID_LICENSE_FEATURE_MAX_COUNT (C macro), 436
NVML_GRID_LICENSE_STATE_LICENSED (C macro), 446
NVML_GRID_LICENSE_STATE_UNINITIALIZED (C macro), 446
NVML_GRID_LICENSE_STATE_UNKNOWN (C macro), 446
NVML_GRID_LICENSE_STATE_UNLICENSED (C macro), 446
NVML_GRID_LICENSE_STATE_UNLICENSED_RESTRICTED (C macro), 446
NVML_GRID_LICENSE_STATE_UNLICENSED_UNRESTRICTED (C macro), 446
NVML_GSP_FIRMWARE_VERSION_BUF_SIZE (C macro), 79
NVML_INIT_FLAG_FORCE_INIT (C macro), 311
NVML_INIT_FLAG_NO_ATTACH (C macro), 311
NVML_INIT_FLAG_NO_GPUS (C macro), 311
NVML_INVALID_VGPU_PLACEMENT_ID (C macro), 436
NVML_MAX_GPU_PERF_PSTATES (C macro), 79
NVML_MAX_GPU_UTILIZATIONS (C macro), 79

NVML_MAX_PHYSICAL_BRIDGE (*C macro*), 222
NVML_MAX_THERMAL_SENSORS_PER_GPU (*C macro*), 222
NVML_MCLK_SWITCH_TYPE_DEFERRED (*C macro*), 307
NVML_MCLK_SWITCH_TYPE_NOT_SUPPORTED (*C macro*), 308
NVML_MCLK_SWITCH_TYPE_RUNTIME (*C macro*), 308
NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_NO_WAIT (*C macro*), 310
NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_TIMEOUT (*C macro*), 310
NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_INFINITE (*C macro*), 310
NVML_NVFBC_SESSION_FLAG_CLASSIFICATIONMAP_ENABLED (*C macro*), 310
NVML_NVFBC_SESSION_FLAG_DIFFMAP_ENABLED (*C macro*), 310
NVML_NVLINK_BER_EXP_SHIFT (*C macro*), 378
NVML_NVLINK_BER_EXP_WIDTH (*C macro*), 378
NVML_NVLINK_BER_MANTISSA_SHIFT (*C macro*), 378
NVML_NVLINK_BER_MANTISSA_WIDTH (*C macro*), 378
NVML_NVLINK_ERROR_COUNTER_BER_GET (*C macro*), 378
NVML_NVLINK_FIRMWARE_UCODE_TYPE_MSE (*C macro*), 378
NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR (*C macro*), 378
NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_CLN (*C macro*), 378
NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_DLN (*C macro*), 378
NVML_NVLINK_FIRMWARE_UCODE_TYPE_NETIR_UPHY (*C macro*), 379
NVML_NVLINK_FIRMWARE_VERSION_LENGTH (*C macro*), 379
NVML_NVLINK_LOW_POWER_THRESHOLD_DEFAULT (*C macro*), 308
NVML_NVLINK_LOW_POWER_THRESHOLD_MAX (*C macro*), 308
NVML_NVLINK_LOW_POWER_THRESHOLD_MIN (*C macro*), 308
NVML_NVLINK_LOW_POWER_THRESHOLD_RESET (*C macro*), 308
NVML_NVLINK_LOW_POWER_THRESHOLD_UNIT_50US (*C macro*), 308
NVML_NVLINK_LOW_POWER_THRESHOLD_UNIT_100US (*C macro*), 308
NVML_NVLINK_MAX_LINKS (*C macro*), 222
NVML_NVLINK_MAX_REMAP_TABLE_ENTRIES_V2 (*C macro*), 379
NVML_NVLINK_NVLE_AES_256_GCM_KEY_SIZE_BYTES (*C macro*), 379
NVML_NVLINK_NVLE_PRIVATE_INFO_SIZE (*C macro*), 379
NVML_NVLINK_POWER_STATE_HIGH_SPEED (*C macro*), 308
NVML_NVLINK_POWER_STATE_LOW (*C macro*), 308
NVML_NVLINK_REMAP_TABLE_ENTRIES_CHUNK (*C macro*), 379
NVML_NVLINK_STATE_ACTIVE (*C macro*), 379
NVML_NVLINK_STATE_ACTIVE_TRAFFIC_DISABLED (*C macro*), 379
NVML_NVLINK_STATE_INACTIVE (*C macro*), 379
NVML_NVLINK_STATE_SLEEP (*C macro*), 379
NVML_NVLINK_TOTAL_SUPPORTED_BW_MODES (*C macro*), 379
NVML_OPERATIONAL_EVENT_ATTR_COMPONENT_RESET (*C macro*), 244
NVML_OPERATIONAL_EVENT_ATTR_LATENT (*C macro*), 244
NVML_OPERATIONAL_EVENT_ATTR_OVERFLOW (*C macro*), 244
NVML_OPERATIONAL_EVENT_ATTR_PRIMARY (*C macro*), 244
NVML_OPERATIONAL_EVENT_ATTR_PROPAGATED (*C macro*), 244
NVML_OPERATIONAL_EVENT_ATTR_THRESHOLD_EXCEEDED (*C macro*), 245
NVML_OPERATIONAL_EVENT_ATTR_UNCONTAINED (*C macro*), 245
NVML_OPERATIONAL_EVENT_GROUP_ATTR_PREVERR (*C macro*), 245
NVML_OPERATIONAL_EVENT_GROUP_ATTR_RECOVERED (*C macro*), 245
NVML_OPERATIONAL_EVENT_GROUP_ATTR_SIMULATED (*C macro*), 245
NVML_PCIE_ATOMICS_CAP_CAS32 (*C macro*), 79
NVML_PCIE_ATOMICS_CAP_CAS64 (*C macro*), 79
NVML_PCIE_ATOMICS_CAP_CAS128 (*C macro*), 79
NVML_PCIE_ATOMICS_CAP_FETCHADD32 (*C macro*), 79
NVML_PCIE_ATOMICS_CAP_FETCHADD64 (*C macro*), 79
NVML_PCIE_ATOMICS_CAP_SWAP32 (*C macro*), 80
NVML_PCIE_ATOMICS_CAP_SWAP64 (*C macro*), 80
NVML_PCIE_ATOMICS_OPS_MAX (*C macro*), 80
NVML_PCIE_LINK_MAX_SPEED_2500MBPS (*C macro*), 80
NVML_PCIE_LINK_MAX_SPEED_5000MBPS (*C macro*), 80
NVML_PCIE_LINK_MAX_SPEED_8000MBPS (*C macro*), 80
NVML_PCIE_LINK_MAX_SPEED_16000MBPS (*C macro*), 80

NVML_PCIE_LINK_MAX_SPEED_32000MBPS (C macro), 80
NVML_PCIE_LINK_MAX_SPEED_64000MBPS (C macro), 80
NVML_PCIE_LINK_MAX_SPEED_INVALID (C macro), 80
NVML_PERF_CF_PM_SENSOR_MAX_SIGNALS (C macro), 222
NVML_PERF_METRICS_CONTROLLER_DLPPC_2X_PWR_POLICY_RELATIONSHIP_SET_LIMITS_MAX (C macro), 222
NVML_PERF_METRICS_CONTROLLER_SAMPLE_CONTROLLER_MAX_NUM (C macro), 222
NVML_PERF_METRICS_CONTROLLER_STATUS_DLPPC_2X_DRAM_CLK_NUM (C macro), 222
NVML_PERF_METRICS_CONTROLLER_TYPE_DLPPC_2X (C macro), 222
NVML_PERF_METRICS_CONTROLLER_TYPE_PFPP_1X (C macro), 222
NVML_PERF_METRICS_NNE_DESC_INFERENCE_LOOPS_MAX (C macro), 222
NVML_PERF_METRICS_PWR_MODEL_DLPPM_1X_MAX_CORE_PMIDATA_MAX_SIZE (C macro), 400
NVML_PERF_METRICS_PWR_MODEL_METRICS_DLPPM_1X_OBSERVED_LINK_DRAM_ESTIMATES_MAX (C macro), 222
NVML_PERF_METRICS_PWR_MODEL_SCALE_LOOPS_MAX_PERF_POLICY_1X (C macro), 223
NVML_PERF_METRICS_PWR_MODEL_SCALE_METRICS_MAX_PERF_SENSOR_1X (C macro), 223
NVML_PERF_METRICS_PWR_MODEL_SCALE_METRICS_MAX_PERF_TOTAL_POWER (C macro), 223
NVML_PERF_METRICS_SAMPLE_COUNT (C macro), 223
NVML_PERF_MODES_BUFFER_SIZE (C macro), 80
NVML_POWER_MIZER_MODE_ADAPTIVE (C macro), 80
NVML_POWER_MIZER_MODE_AUTO (C macro), 80
NVML_POWER_MIZER_MODE_PREFER_CONSISTENT_PERFORMANCE (C macro), 80
NVML_POWER_MIZER_MODE_PREFER_MAXIMUM_PERFORMANCE (C macro), 81
NVML_POWER_SCOPE_COUNT (C macro), 81
NVML_POWER_SCOPE_GPU (C macro), 81
NVML_POWER_SCOPE_GPU_BASE (C macro), 81
NVML_POWER_SCOPE_MEMORY (C macro), 81
NVML_POWER_SCOPE_MODULE (C macro), 81
NVML_POWER_SMOOTHING_ADMIN_OVERRIDE_NOT_SET (C macro), 411
NVML_POWER_SMOOTHING_IDX_FROM_FIELD_VAL (C macro), 411
NVML_POWER_SMOOTHING_IDX_FROM_FIELD_VAL_DEPRECATED (C macro), 411
NVML_POWER_SMOOTHING_MAX_NUM_PROFILES (C macro), 411
NVML_POWER_SMOOTHING_NUM_PROFILE_PARAMS (C macro), 412
NVML_POWER_SMOOTHING_PROFILE_PARAM_PERCENT_TMP_FLOOR_AC (C macro), 412
NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_AC (C macro), 412
NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_AC (C macro), 412
NVML_POWER_SMOOTHING_PROFILE_PARAM_PRIMARY_FLOOR_TA (C macro), 412
NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_DOWN_HYSTER (C macro), 412
NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_DOWN_RATE (C macro), 412
NVML_POWER_SMOOTHING_PROFILE_PARAM_RAMP_UP_RATE (C macro), 412
NVML_POWER_SMOOTHING_PROFILE_PARAM_SECONDARY_POWER_ (C macro), 412
NVML_POWER_SOURCE_AC (C macro), 81
NVML_POWER_SOURCE_BATTERY (C macro), 81
NVML_POWER_SOURCE_UNDERSIZED (C macro), 81
NVML_RUSD_POLL_ALL (C macro), 81
NVML_RUSD_POLL_FAN (C macro), 81
NVML_RUSD_POLL_MEMORY (C macro), 81
NVML_RUSD_POLL_NONE (C macro), 81
NVML_RUSD_POLL_PCI (C macro), 82
NVML_RUSD_POLL_PERF (C macro), 82
NVML_RUSD_POLL_POWER (C macro), 82
NVML_RUSD_POLL_PROC_UTIL (C macro), 82
NVML_RUSD_POLL_THERMAL (C macro), 82
NVML_SCHEDULER_SW_MAX_LOG_ENTRIES (C macro), 446
NVML_SINGLE_BIT_ECC (C macro), 82
NVML_SUPPORTED_VGPU_SCHEDULER_POLICY_COUNT (C macro), 446
NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE (C macro), 53
NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE (C macro), 53
NVML_TOPOLOGY_CPU (C macro), 223
NVML_VALUE_NOT_AVAILABLE (C macro), 223
NVML_VGPU_NAME_BUFFER_SIZE (C macro), 436
NVML_VGPU_PGPU_HETEROGENEOUS_MODE (C macro), 436
NVML_VGPU_PGPU_HOMOGENEOUS_MODE (C macro), 436
NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION (C macro), 436
NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION_NO (C macro), 436
NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION_YES (C macro), 436
NVML_VGPU_SCHEDULER_ARR_DEFAULT (C

macro), 446
NVML_VGPU_SCHEDULER_ARR_DISABLE (*C macro*), 446
NVML_VGPU_SCHEDULER_ARR_ENABLE (*C macro*), 446
NVML_VGPU_SCHEDULER_ENGINE_TYPE_GRAPHICS (*C macro*), 446
NVML_VGPU_SCHEDULER_ENGINE_TYPE_NVENC0 (*C macro*), 446
NVML_VGPU_SCHEDULER_ENGINE_TYPE_NVENC1 (*C macro*), 446
NVML_VGPU_SCHEDULER_POLICY_BEST_EFFORT (*C macro*), 446
NVML_VGPU_SCHEDULER_POLICY_EQUAL_SHARE (*C macro*), 447
NVML_VGPU_SCHEDULER_POLICY_FIXED_SHARE (*C macro*), 447
NVML_VGPU_SCHEDULER_POLICY_UNKNOWN (*C macro*), 447
NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION (*C macro*), 436
NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION_NO (*C macro*), 436
NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION_YES (*C macro*), 436
NVML_WORKLOAD_POWER_MAX_PROFILES (*C macro*), 404
nvmlAccountingStats_t (*C++ struct*), 502
nvmlAccountingStats_t::gpuUtilization (*C++ member*), 503
nvmlAccountingStats_t::isRunning (*C++ member*), 503
nvmlAccountingStats_t::maxMemoryUsage (*C++ member*), 503
nvmlAccountingStats_t::memoryUtilization (*C++ member*), 503
nvmlAccountingStats_t::reserved (*C++ member*), 503
nvmlAccountingStats_t::startTime (*C++ member*), 503
nvmlAccountingStats_t::time (*C++ member*), 503
nvmlAccountingStats_v2_t (*C++ struct*), 503
nvmlAccountingStats_v2_t::gpuUtilization (*C++ member*), 503
nvmlAccountingStats_v2_t::isRunning (*C++ member*), 503
nvmlAccountingStats_v2_t::maxMemoryUsage (*C++ member*), 504
nvmlAccountingStats_v2_t::memoryUtilization (*C++ member*), 504
nvmlAccountingStats_v2_t::pid (*C++ member*), 503
nvmlAccountingStats_v2_t::sampleCount (*C++ member*), 504
nvmlAccountingStats_v2_t::startTime (*C++ member*), 504
nvmlAccountingStats_v2_t::sumFbUtil (*C++ member*), 504
nvmlAccountingStats_v2_t::sumGpuUtil (*C++ member*), 504
nvmlAccountingStats_v2_t::time (*C++ member*), 504
nvmlActiveVgpuInstanceInfo_t (*C++ type*), 449
nvmlActiveVgpuInstanceInfo_v1 (*C macro*), 447
nvmlActiveVgpuInstanceInfo_v1_t (*C++ struct*), 504
nvmlActiveVgpuInstanceInfo_v1_t::version (*C++ member*), 504
nvmlActiveVgpuInstanceInfo_v1_t::vgpuCount (*C++ member*), 504
nvmlActiveVgpuInstanceInfo_v1_t::vgpuInstances (*C++ member*), 504
nvmlAdaptiveTgpModeInfo_v1_t (*C++ struct*), 504
nvmlAdaptiveTgpModeInfo_v1_t::adjustedLimitMw (*C++ member*), 505
nvmlAdaptiveTgpModeInfo_v1_t::adminOverrideEnabled (*C++ member*), 505
nvmlAdaptiveTgpModeInfo_v1_t::enablementStatus (*C++ member*), 505
nvmlAdaptiveTgpModeInfo_v1_t::featureAllowedByAdmi (*C++ member*), 505
nvmlAdaptiveTgpModeInfo_v1_t::inBandEnableRequest (*C++ member*), 505
nvmlAffinityScope_t (*C++ type*), 101
nvmlBAR1Memory_t (*C++ struct*), 505
nvmlBAR1Memory_t::bar1Free (*C++ member*), 505
nvmlBAR1Memory_t::bar1Total (*C++ member*), 505
nvmlBAR1Memory_t::bar1Used (*C++ member*), 505
nvmlBBXTimeData_v1_t (*C++ struct*), 505
nvmlBBXTimeData_v1_t::timeRun (*C++ member*), 505
nvmlBrandType_t (*C++ enum*), 83
nvmlBrandType_t::NVML_BRAND_COUNT (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_GEFORCE (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_GEFORCE_RTX (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_GRID (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA (*C++*

enumerator), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_CLOUD_GAMING (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_DLA (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_NPU (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_RTX (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_VAPPS (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_VCS (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_VGAMEDEV (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_VGAMING (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_VPC (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVIDIA_VWS (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_NVS (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_QUADRO (*C++ enumerator*), 83
nvmlBrandType_t::NVML_BRAND_QUADRO_RTX (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_TESLA (*C++ enumerator*), 83
nvmlBrandType_t::NVML_BRAND_TITAN (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_TITAN_RTX (*C++ enumerator*), 84
nvmlBrandType_t::NVML_BRAND_UNKNOWN (*C++ enumerator*), 83
nvmlBridgeChipHierarchy_t (*C++ struct*), 506
nvmlBridgeChipHierarchy_t::bridgeChipInfo (*C++ member*), 506
nvmlBridgeChipHierarchy_t::bridgeCount (*C++ member*), 506
nvmlBridgeChipInfo_t (*C++ struct*), 506
nvmlBridgeChipInfo_t::fwVersion (*C++ member*), 506
nvmlBridgeChipInfo_t::type (*C++ member*), 506
nvmlBridgeChipType_t (*C++ enum*), 224
nvmlBridgeChipType_t::NVML_BRIDGE_CHIP_BRO4 (*C++ enumerator*), 224
nvmlBridgeChipType_t::NVML_BRIDGE_CHIP_PLX (*C++ enumerator*), 224
nvmlBusType_t (*C++ type*), 94
nvmlC2cModeInfo_v1 (*C macro*), 223
nvmlC2cModeInfo_v1_t (*C++ struct*), 506
nvmlC2cModeInfo_v1_t::isC2cEnabled (*C++ member*), 506
nvmlClkMonFaultInfo_t (*C++ struct*), 507
nvmlClkMonFaultInfo_t::clkApiDomain (*C++ member*), 507
nvmlClkMonFaultInfo_t::clkDomainFaultMask (*C++ member*), 507
nvmlClkMonStatus_t (*C++ struct*), 507
nvmlClkMonStatus_t::bGlobalStatus (*C++ member*), 507
nvmlClkMonStatus_t::clkMonList (*C++ member*), 508
nvmlClkMonStatus_t::clkMonListSize (*C++ member*), 507
nvmlClockId_t (*C++ enum*), 84
nvmlClockId_t::NVML_CLOCK_ID_APP_CLOCK_DEFAULT (*C++ enumerator*), 85
nvmlClockId_t::NVML_CLOCK_ID_APP_CLOCK_TARGET (*C++ enumerator*), 85
nvmlClockId_t::NVML_CLOCK_ID_COUNT (*C++ enumerator*), 85
nvmlClockId_t::NVML_CLOCK_ID_CURRENT (*C++ enumerator*), 85
nvmlClockId_t::NVML_CLOCK_ID_CUSTOMER_BOOST_MAX (*C++ enumerator*), 85
nvmlClockLimitId_t (*C++ enum*), 55
nvmlClockLimitId_t::NVML_CLOCK_LIMIT_ID_RANGE_START (*C++ enumerator*), 55
nvmlClockLimitId_t::NVML_CLOCK_LIMIT_ID_TDP (*C++ enumerator*), 55
nvmlClockLimitId_t::NVML_CLOCK_LIMIT_ID_UNLIMITED (*C++ enumerator*), 55
nvmlClockOffset_t (*C++ type*), 94
nvmlClockOffset_v1 (*C macro*), 82
nvmlClockOffset_v1_t (*C++ struct*), 508
nvmlClockOffset_v1_t::clockOffsetMHz (*C++ member*), 508
nvmlClockOffset_v1_t::maxClockOffsetMHz (*C++ member*), 508
nvmlClockOffset_v1_t::minClockOffsetMHz (*C++ member*), 508
nvmlClockOffset_v1_t::pstate (*C++ member*), 508
nvmlClockOffset_v1_t::type (*C++ member*), 508
nvmlClockOffset_v1_t::version (*C++ member*), 508
nvmlClocksEventReasonAll (*C macro*), 395
nvmlClocksEventReasonApplicationsClocksSetting (*C macro*), 395
nvmlClocksEventReasonBoardLimit (*C macro*), 395
nvmlClocksEventReasonDisplayClockSetting (*C macro*), 395
nvmlClocksEventReasonGpuIdle (*C macro*),

395
nvmlClocksEventReasonNone (*C macro*), 396
nvmlClocksEventReasonReliability (*C macro*), 396
nvmlClocksEventReasonSwPowerCap (*C macro*), 396
nvmlClocksEventReasonSwThermalSlowdown (*C macro*), 396
nvmlClocksEventReasonSyncBoost (*C macro*), 396
nvmlClocksThrottleReasonAll (*C macro*), 396
nvmlClocksThrottleReasonApplicationsClocksSetting (*C macro*), 397
nvmlClocksThrottleReasonDisplayClockSetting (*C macro*), 397
nvmlClocksThrottleReasonGpuIdle (*C macro*), 397
nvmlClocksThrottleReasonHwPowerBrakeSlowdown (*C macro*), 397
nvmlClocksThrottleReasonHwSlowdown (*C macro*), 397
nvmlClocksThrottleReasonHwThermalSlowdown (*C macro*), 398
nvmlClocksThrottleReasonNone (*C macro*), 398
nvmlClocksThrottleReasonSwPowerCap (*C macro*), 399
nvmlClocksThrottleReasonSwThermalSlowdown (*C macro*), 399
nvmlClocksThrottleReasonSyncBoost (*C macro*), 399
nvmlClocksThrottleReasonUserDefinedClocks (*C macro*), 399
nvmlClockType_t (*C++ enum*), 85
nvmlClockType_t::NVML_CLOCK_COUNT (*C++ enumerator*), 85
nvmlClockType_t::NVML_CLOCK_GRAPHICS (*C++ enumerator*), 85
nvmlClockType_t::NVML_CLOCK_MEM (*C++ enumerator*), 85
nvmlClockType_t::NVML_CLOCK_SM (*C++ enumerator*), 85
nvmlClockType_t::NVML_CLOCK_VIDEO (*C++ enumerator*), 85
nvmlComputeInstance_t (*C++ type*), 338
nvmlComputeInstanceDestroy (*C++ function*), 322
nvmlComputeInstanceGetInfo_v2 (*C++ function*), 322
nvmlComputeInstanceInfo_t (*C++ struct*), 508
nvmlComputeInstanceInfo_t::device (*C++ member*), 508
nvmlComputeInstanceInfo_t::gpuInstance (*C++ member*), 508
nvmlComputeInstanceInfo_t::id (*C++ member*), 508
nvmlComputeInstanceInfo_t::placement (*C++ member*), 508
nvmlComputeInstanceInfo_t::profileId (*C++ member*), 508
nvmlComputeInstancePlacement_t (*C++ struct*), 509
nvmlComputeInstancePlacement_t::size (*C++ member*), 509
nvmlComputeInstancePlacement_t::start (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t (*C++ struct*), 509
nvmlComputeInstanceProfileInfo_t::id (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t::instanceCount (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t::multiprocessorCou (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t::sharedCopyEngineC (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t::sharedDecoderCour (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t::sharedEncoderCour (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t::sharedJpegCount (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t::sharedOfaCount (*C++ member*), 509
nvmlComputeInstanceProfileInfo_t::sliceCount (*C++ member*), 509
nvmlComputeInstanceProfileInfo_v2 (*C macro*), 321
nvmlComputeInstanceProfileInfo_v2_t (*C++ struct*), 510
nvmlComputeInstanceProfileInfo_v2_t::id (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::instanceCount (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::multiprocesso (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::name (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::sharedCopyEngi (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::sharedDecoderC (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::sharedEncoderC (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::sharedJpegCour (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::sharedOfaCount (*C++ member*), 510

nvmlComputeInstanceProfileInfo_v2_t::sliceCount (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v2_t::version (*C++ member*), 510
nvmlComputeInstanceProfileInfo_v3 (*C macro*), 321
nvmlComputeInstanceProfileInfo_v3_t (*C++ struct*), 511
nvmlComputeInstanceProfileInfo_v3_t::capabilities (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::id (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::instanceCount (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::multiprocessorCount (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::name (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::sharedCopyEngineCount (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::sharedDecoderCount (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::sharedEncoderCount (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::sharedJpegCount (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::sharedOfaCount (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::sliceCount (*C++ member*), 511
nvmlComputeInstanceProfileInfo_v3_t::version (*C++ member*), 511
nvmlComputeMode_t (*C++ enum*), 85
nvmlComputeMode_t::NVML_COMPUTEMODE_COUNT (*C++ enumerator*), 86
nvmlComputeMode_t::NVML_COMPUTEMODE_DEFAULT (*C++ enumerator*), 86
nvmlComputeMode_t::NVML_COMPUTEMODE_EXCLUSIVE_PROCESS (*C++ enumerator*), 86
nvmlComputeMode_t::NVML_COMPUTEMODE_EXCLUSIVE_THREAD (*C++ enumerator*), 86
nvmlComputeMode_t::NVML_COMPUTEMODE_PROHIBITED (*C++ enumerator*), 86
nvmlConfComputeGetKeyRotationThresholdInfo_t (*C++ type*), 52
nvmlConfComputeGetKeyRotationThresholdInfo_v1 (*C macro*), 52
nvmlConfComputeGetKeyRotationThresholdInfo_v1_t (*C++ struct*), 512
nvmlConfComputeGetKeyRotationThresholdInfo_v1_t::maxAttackerAdvantage (*C++ member*), 512
nvmlConfComputeGetKeyRotationThresholdInfo_v1_t::version (*C++ member*), 512
nvmlConfComputeGpuAttestationReport_t (*C++ struct*), 512
nvmlConfComputeGpuAttestationReport_t::attestationReportBuffer (*C++ member*), 512
nvmlConfComputeGpuAttestationReport_t::attestationReportSize (*C++ member*), 512
nvmlConfComputeGpuAttestationReport_t::cecAttestationDataBuffer (*C++ member*), 512
nvmlConfComputeGpuAttestationReport_t::cecAttestationDataSize (*C++ member*), 512
nvmlConfComputeGpuAttestationReport_t::isCecAttestationReportPresent (*C++ member*), 512
nvmlConfComputeGpuAttestationReport_t::nonce (*C++ member*), 512
nvmlConfComputeGpuCertificate_t (*C++ struct*), 512
nvmlConfComputeGpuCertificate_t::attestationCertChain (*C++ member*), 513
nvmlConfComputeGpuCertificate_t::attestationCertChainSize (*C++ member*), 512
nvmlConfComputeGpuCertificate_t::certChain (*C++ member*), 512
nvmlConfComputeGpuCertificate_t::certChainSize (*C++ member*), 512
nvmlConfComputeMemSizeInfo_t (*C++ struct*), 513
nvmlConfComputeMemSizeInfo_t::protectedMemSizeKib (*C++ member*), 513
nvmlConfComputeMemSizeInfo_t::unprotectedMemSizeKib (*C++ member*), 513
nvmlConfComputeSetKeyRotationThresholdInfo_t (*C++ type*), 52
nvmlConfComputeSetKeyRotationThresholdInfo_v1 (*C macro*), 52
nvmlConfComputeSetKeyRotationThresholdInfo_v1_t (*C++ struct*), 513
nvmlConfComputeSetKeyRotationThresholdInfo_v1_t::maxAttackerAdvantage (*C++ member*), 513
nvmlConfComputeSetKeyRotationThresholdInfo_v1_t::version (*C++ member*), 513
nvmlConfComputeSystemCaps_t (*C++ struct*), 513
nvmlConfComputeSystemCaps_t::cpuCaps (*C++ member*), 513
nvmlConfComputeSystemCaps_t::gpusCaps (*C++ member*), 513
nvmlConfComputeSystemState_t (*C++ struct*), 513
nvmlConfComputeSystemState_t::ccFeature (*C++ member*), 514
nvmlConfComputeSystemState_t::devToolsMode (*C++ member*), 514
nvmlConfComputeSystemState_t::environment (*C++ member*), 514
nvmlCoolerControl_t (*C++ enum*), 224

nvmlCoolerControl_t::NVML_THERMAL_COOLER_SIGNAL_COUNT (C++ enumerator), 224
nvmlCoolerControl_t::NVML_THERMAL_COOLER_SIGNAL_NONE (C++ enumerator), 224
nvmlCoolerControl_t::NVML_THERMAL_COOLER_SIGNAL_TOGGLE (C++ enumerator), 224
nvmlCoolerControl_t::NVML_THERMAL_COOLER_SIGNAL_VARIABLE (C++ enumerator), 224
nvmlCoolerInfo_t (C++ type), 233
nvmlCoolerInfo_v1 (C macro), 223
nvmlCoolerInfo_v1_t (C++ struct), 514
nvmlCoolerInfo_v1_t::index (C++ member), 514
nvmlCoolerInfo_v1_t::signalType (C++ member), 514
nvmlCoolerInfo_v1_t::target (C++ member), 514
nvmlCoolerInfo_v1_t::version (C++ member), 514
nvmlCoolerTarget_t (C++ enum), 224
nvmlCoolerTarget_t::NVML_THERMAL_COOLER_TARGET_GPU (C++ enumerator), 224
nvmlCoolerTarget_t::NVML_THERMAL_COOLER_TARGET_GPU_RELATED (C++ enumerator), 225
nvmlCoolerTarget_t::NVML_THERMAL_COOLER_TARGET_MEMORY (C++ enumerator), 224
nvmlCoolerTarget_t::NVML_THERMAL_COOLER_TARGET_NONE (C++ enumerator), 224
nvmlCoolerTarget_t::NVML_THERMAL_COOLER_TARGET_POWER_SUPPLY (C++ enumerator), 224
nvmlCoreRailMetrics_t (C++ struct), 514
nvmlCoreRailMetrics_t::rails (C++ member), 514
nvmlCPERCursor_v1_t (C++ struct), 506
nvmlCPERCursor_v1_t::cperTypeMask (C++ member), 507
nvmlCPERCursor_v1_t::handle (C++ member), 507
nvmlCPERCursor_v1_t::uuid (C++ member), 507
nvmlCPERCursorHandle_t (C++ type), 46
nvmlCPERType_t (C++ enum), 46
nvmlCPERType_t::NVML_CPER_ACCESS_TYPE_GPU (C++ enumerator), 46
nvmlDetachGpuState_t (C++ enum), 234
nvmlDetachGpuState_t::NVML_DETACH_GPU_KEEP (C++ enumerator), 234
nvmlDetachGpuState_t::NVML_DETACH_GPU_REMOVE (C++ enumerator), 234
nvmlDevice_t (C++ type), 233
nvmlDeviceAddressingMode_t (C++ type), 233
nvmlDeviceAddressingMode_v1 (C macro), 223
nvmlDeviceAddressingMode_v1_t (C++ struct), 514
nvmlDeviceAddressingMode_v1_t::value (C++ member), 514
nvmlDeviceAddressingMode_v1_t::version (C++ member), 514
nvmlDeviceAddressingModeType_t (C++ enum), 225
nvmlDeviceAddressingModeType_t::NVML_DEVICE_ADDRESSING_MODE_HMM (C++ enumerator), 225
nvmlDeviceAddressingModeType_t::NVML_DEVICE_ADDRESSING_MODE_NONE (C++ enumerator), 225
nvmlDeviceAddressingModeType_t::NVML_DEVICE_ADDRESSING_MODE_PASID (C++ enumerator), 225
nvmlDeviceArchitecture_t (C++ type), 94
nvmlDeviceAttributes_t (C++ struct), 515
nvmlDeviceAttributes_t::computeInstanceSliceCount (C++ member), 515
nvmlDeviceAttributes_t::gpuInstanceSliceCount (C++ member), 515
nvmlDeviceAttributes_t::memorySizeMB (C++ member), 515
nvmlDeviceAttributes_t::multiprocessorCount (C++ member), 515
nvmlDeviceAttributes_t::sharedCopyEngineCount (C++ member), 515
nvmlDeviceAttributes_t::sharedDecoderCount (C++ member), 515
nvmlDeviceAttributes_t::sharedEncoderCount (C++ member), 515
nvmlDeviceAttributes_t::sharedJpegCount (C++ member), 515
nvmlDeviceAttributes_t::sharedOfaCount (C++ member), 515
nvmlDeviceCapabilities_v1_t (C++ struct), 515
nvmlDeviceCapabilities_v1_t::capMask (C++ member), 515
nvmlDeviceCapabilities_v1_t::version (C++ member), 515
nvmlDeviceClearAccountingPids (C++ function), 39
nvmlDeviceClearCpuAffinity (C++ function), 96
nvmlDeviceClearEccErrorCounts (C++ function), 55
nvmlDeviceClearFieldValues (C++ function), 308
nvmlDeviceCreateGpuInstance (C++ function), 323
nvmlDeviceCreateGpuInstanceWithPlacement (C++ function), 323
nvmlDeviceCurrentClockFreqs_t (C++ type), 94
nvmlDeviceCurrentClockFreqs_v1 (C macro), 82

nvmlDeviceCurrentClockFreqs_v1_t (*C++ struct*), 516
nvmlDeviceCurrentClockFreqs_v1_t::str (*C++ member*), 516
nvmlDeviceCurrentClockFreqs_v1_t::version (*C++ member*), 516
nvmlDeviceDiscoverGpus (*C++ function*), 235
nvmlDeviceFreezeNvLinkUtilizationCounter (*C++ function*), 381
nvmlDeviceGetAccountingBufferSize (*C++ function*), 40
nvmlDeviceGetAccountingMode (*C++ function*), 40
nvmlDeviceGetAccountingPids (*C++ function*), 41
nvmlDeviceGetAccountingStats (*C++ function*), 42
nvmlDeviceGetAccountingStats_v2 (*C++ function*), 43
nvmlDeviceGetActiveVgpus (*C++ function*), 453
nvmlDeviceGetAdaptiveClockInfoStatus (*C++ function*), 113
nvmlDeviceGetAdaptiveTgpModeInfo_v1 (*C++ function*), 113
nvmlDeviceGetAddressingMode (*C++ function*), 96
nvmlDeviceGetAPIRestriction (*C++ function*), 112
nvmlDeviceGetApplicationsClock (*C++ function*), 114
nvmlDeviceGetArchitecture (*C++ function*), 114
nvmlDeviceGetAttributes_v2 (*C++ function*), 114
nvmlDeviceGetAutoBoostedClocksEnabled (*C++ function*), 115
nvmlDeviceGetBankRemapperStatus_v1 (*C++ function*), 117
nvmlDeviceGetBAR1MemoryInfo (*C++ function*), 116
nvmlDeviceGetBBXTimeData_v1 (*C++ function*), 116
nvmlDeviceGetBoardId (*C++ function*), 117
nvmlDeviceGetBoardPartNumber (*C++ function*), 118
nvmlDeviceGetBrand (*C++ function*), 118
nvmlDeviceGetBridgeChipInfo (*C++ function*), 119
nvmlDeviceGetBusType (*C++ function*), 119
nvmlDeviceGetC2cModeInfoV (*C++ function*), 120
nvmlDeviceGetClkMonStatus (*C++ function*), 120
nvmlDeviceGetClock (*C++ function*), 121
nvmlDeviceGetClockInfo (*C++ function*), 121
nvmlDeviceGetClockOffsets (*C++ function*), 122
nvmlDeviceGetComputeInstanceId (*C++ function*), 324
nvmlDeviceGetComputeMode (*C++ function*), 122
nvmlDeviceGetComputeRunningProcesses_v3 (*C++ function*), 123
nvmlDeviceGetConfComputeGpuAttestationReport (*C++ function*), 124
nvmlDeviceGetConfComputeGpuCertificate (*C++ function*), 124
nvmlDeviceGetConfComputeMemSizeInfo (*C++ function*), 125
nvmlDeviceGetConfComputeProtectedMemoryUsage (*C++ function*), 125
nvmlDeviceGetCoolerInfo (*C++ function*), 126
nvmlDeviceGetCount_v2 (*C++ function*), 126
nvmlDeviceGetCpuAffinity (*C++ function*), 97
nvmlDeviceGetCpuAffinityWithinScope (*C++ function*), 98
nvmlDeviceGetCreatableVgpus (*C++ function*), 453
nvmlDeviceGetCudaComputeCapability (*C++ function*), 127
nvmlDeviceGetCurrentClockFreqs (*C++ function*), 128
nvmlDeviceGetCurrentClocksEventReasons (*C++ function*), 129
nvmlDeviceGetCurrentClocksThrottleReasons (*C++ function*), 130
nvmlDeviceGetCurrPcieLinkGeneration (*C++ function*), 128
nvmlDeviceGetCurrPcieLinkWidth (*C++ function*), 128
nvmlDeviceGetDecoderUtilization (*C++ function*), 130
nvmlDeviceGetDefaultApplicationsClock (*C++ function*), 131
nvmlDeviceGetDefaultEccMode (*C++ function*), 131
nvmlDeviceGetDetailedEccErrors (*C++ function*), 132
nvmlDeviceGetDeviceHandleFromMigDeviceHandle (*C++ function*), 324
nvmlDeviceGetDisplayActive (*C++ function*), 133
nvmlDeviceGetDisplayMode (*C++ function*), 133
nvmlDeviceGetDramEncryptionMode (*C++ function*), 134
nvmlDeviceGetDriverModel_v2 (*C++ func-*

tion), 134
nvmlDeviceGetDynamicPstatesInfo (*C++ function*), 135
nvmlDeviceGetEccMode (*C++ function*), 136
nvmlDeviceGetEncoderCapacity (*C++ function*), 136
nvmlDeviceGetEncoderSessions (*C++ function*), 137
nvmlDeviceGetEncoderStats (*C++ function*), 138
nvmlDeviceGetEncoderUtilization (*C++ function*), 138
nvmlDeviceGetEnforcedPowerLimit (*C++ function*), 139
nvmlDeviceGetFanControlPolicy_v2 (*C++ function*), 141
nvmlDeviceGetFanSpeed (*C++ function*), 141
nvmlDeviceGetFanSpeed_v2 (*C++ function*), 142
nvmlDeviceGetFanSpeedRPM (*C++ function*), 142
nvmlDeviceGetFBCSessions (*C++ function*), 139
nvmlDeviceGetFBCStats (*C++ function*), 140
nvmlDeviceGetFieldValues (*C++ function*), 309
nvmlDeviceGetGpcClkMinMaxVfOffset (*C++ function*), 143
nvmlDeviceGetGpcClkVfOffset (*C++ function*), 143
nvmlDeviceGetGpuFabricInfo (*C++ function*), 143
nvmlDeviceGetGpuFabricInfo_v4 (*C++ function*), 145
nvmlDeviceGetGpuFabricInfoV (*C++ function*), 144
nvmlDeviceGetGpuInstanceById (*C++ function*), 325
nvmlDeviceGetGpuInstanceId (*C++ function*), 325
nvmlDeviceGetGpuInstancePossiblePlacements_v2 (*C++ function*), 326
nvmlDeviceGetGpuInstanceProfileInfo (*C++ function*), 326
nvmlDeviceGetGpuInstanceProfileInfoByIdV (*C++ function*), 327
nvmlDeviceGetGpuInstanceProfileInfoV (*C++ function*), 328
nvmlDeviceGetGpuInstanceRemainingCapacity (*C++ function*), 329
nvmlDeviceGetGpuInstances (*C++ function*), 329
nvmlDeviceGetGpuMaxPcieLinkGeneration (*C++ function*), 145
nvmlDeviceGetGpuOperationMode (*C++ function*), 146
nvmlDeviceGetGpuUptime_v1 (*C++ function*), 146
nvmlDeviceGetGraphicsRunningProcesses_v3 (*C++ function*), 147
nvmlDeviceGetGridLicensableFeatures_v4 (*C++ function*), 427
nvmlDeviceGetGspFirmwareMode (*C++ function*), 148
nvmlDeviceGetGspFirmwareVersion (*C++ function*), 148
nvmlDeviceGetHandleByIndex_v2 (*C++ function*), 149
nvmlDeviceGetHandleByPciBusId_v2 (*C++ function*), 150
nvmlDeviceGetHandleBySerial (*C++ function*), 151
nvmlDeviceGetHandleByUUID (*C++ function*), 152
nvmlDeviceGetHandleByUUIDV (*C++ function*), 153
nvmlDeviceGetHostname_v1 (*C++ function*), 154
nvmlDeviceGetHostVgpuMode (*C++ function*), 427
nvmlDeviceGetIndex (*C++ function*), 154
nvmlDeviceGetInforomConfigurationChecksum (*C++ function*), 155
nvmlDeviceGetInforomImageVersion (*C++ function*), 156
nvmlDeviceGetInforomVersion (*C++ function*), 157
nvmlDeviceGetIrqNum (*C++ function*), 158
nvmlDeviceGetJpgUtilization (*C++ function*), 158
nvmlDeviceGetLastBBXFlushTime (*C++ function*), 159
nvmlDeviceGetMarginTemperature (*C++ function*), 160
nvmlDeviceGetMaxClockInfo (*C++ function*), 161
nvmlDeviceGetMaxCustomerBoostClock (*C++ function*), 161
nvmlDeviceGetMaxMigDeviceCount (*C++ function*), 330
nvmlDeviceGetMaxPcieLinkGeneration (*C++ function*), 162
nvmlDeviceGetMaxPcieLinkWidth (*C++ function*), 162
nvmlDeviceGetMemClkMinMaxVfOffset (*C++ function*), 163
nvmlDeviceGetMemClkVfOffset (*C++ function*), 163

nvmlDeviceGetMemoryAffinity (*C++ function*), 99
nvmlDeviceGetMemoryBusWidth (*C++ function*), 164
nvmlDeviceGetMemoryErrorCounter (*C++ function*), 164
nvmlDeviceGetMemoryInfo (*C++ function*), 165
nvmlDeviceGetMemoryInfo_v2 (*C++ function*), 166
nvmlDeviceGetMemoryLimits_v1 (*C++ function*), 168
nvmlDeviceGetMigDeviceHandleByIndex (*C++ function*), 330
nvmlDeviceGetMigMode (*C++ function*), 331
nvmlDeviceGetMinMaxClockOfPState (*C++ function*), 168
nvmlDeviceGetMinMaxFanSpeed (*C++ function*), 169
nvmlDeviceGetMinorNumber (*C++ function*), 169
nvmlDeviceGetModuleId (*C++ function*), 170
nvmlDeviceGetMPSComputeRunningProcesses_v3 (*C++ function*), 159
nvmlDeviceGetMultiGpuBoard (*C++ function*), 170
nvmlDeviceGetName (*C++ function*), 171
nvmlDeviceGetNumaNodeId (*C++ function*), 99
nvmlDeviceGetNumFans (*C++ function*), 171
nvmlDeviceGetNumGpuCores (*C++ function*), 172
nvmlDeviceGetNvlinkBwMode (*C++ function*), 387
nvmlDeviceGetNvLinkCapability (*C++ function*), 381
nvmlDeviceGetNvLinkErrorCounter (*C++ function*), 382
nvmlDeviceGetNvLinkInfo (*C++ function*), 382
nvmlDeviceGetNvLinkInfo_v3 (*C++ function*), 383
nvmlDeviceGetNvlinkNvlePrivateInfo_v1 (*C++ function*), 387
nvmlDeviceGetNvlinkRemapTableInfo_v1 (*C++ function*), 388
nvmlDeviceGetNvLinkRemoteDeviceType (*C++ function*), 383
nvmlDeviceGetNvLinkRemotePciInfo_v2 (*C++ function*), 384
nvmlDeviceGetNvLinkState (*C++ function*), 384
nvmlDeviceGetNvlinkSupportedBwModes (*C++ function*), 388
nvmlDeviceGetNvLinkTelemetrySamples_v1 (*C++ function*), 385
nvmlDeviceGetNvLinkUtilizationControl (*C++ function*), 385
nvmlDeviceGetNvLinkUtilizationCounter (*C++ function*), 386
nvmlDeviceGetNvLinkVersion (*C++ function*), 386
nvmlDeviceGetOfaUtilization (*C++ function*), 172
nvmlDeviceGetP2PStatus (*C++ function*), 173
nvmlDeviceGetPcieLinkMaxSpeed (*C++ function*), 175
nvmlDeviceGetPcieReplayCounter (*C++ function*), 175
nvmlDeviceGetPcieSpeed (*C++ function*), 176
nvmlDeviceGetPcieThroughput (*C++ function*), 176
nvmlDeviceGetPciInfo_v3 (*C++ function*), 174
nvmlDeviceGetPciInfoExt (*C++ function*), 174
nvmlDeviceGetPdi (*C++ function*), 177
nvmlDeviceGetPerformanceModes (*C++ function*), 177
nvmlDeviceGetPerformanceState (*C++ function*), 178
nvmlDeviceGetPersistenceMode (*C++ function*), 178
nvmlDeviceGetPgpuMetadataString (*C++ function*), 482
nvmlDeviceGetPlatformInfo (*C++ function*), 179
nvmlDeviceGetPowerManagementDefaultLimit (*C++ function*), 180
nvmlDeviceGetPowerManagementLimit (*C++ function*), 180
nvmlDeviceGetPowerManagementLimitConstraints (*C++ function*), 181
nvmlDeviceGetPowerManagementMode (*C++ function*), 181
nvmlDeviceGetPowerMizerMode_v1 (*C++ function*), 182
nvmlDeviceGetPowerSource (*C++ function*), 183
nvmlDeviceGetPowerState (*C++ function*), 183
nvmlDeviceGetPowerUsage (*C++ function*), 183
nvmlDeviceGetProcessesUtilizationInfo (*C++ function*), 185
nvmlDeviceGetProcessUtilization (*C++ function*), 184
nvmlDeviceGetRemappedRows (*C++ function*), 187
nvmlDeviceGetRemappedRows_v2 (*C++ function*), 187
nvmlDeviceGetRepairStatus (*C++ function*), 100
nvmlDeviceGetRetiredPages (*C++ function*), 188

nvmlDeviceGetRetiredPages_v2 (*C++ function*), 189
nvmlDeviceGetRetiredPagesPendingStatus (*C++ function*), 189
nvmlDeviceGetRowRemapperHistogram (*C++ function*), 190
nvmlDeviceGetRunningProcessDetailList (*C++ function*), 190
nvmlDeviceGetSamples (*C++ function*), 191
nvmlDeviceGetSerial (*C++ function*), 193
nvmlDeviceGetSramEccErrorStatus (*C++ function*), 193
nvmlDeviceGetSramUniqueUncorrectedEccErrorCounts (*C++ function*), 194
nvmlDeviceGetSupportedClocksEventReasons (*C++ function*), 195
nvmlDeviceGetSupportedClocksThrottleReasons (*C++ function*), 195
nvmlDeviceGetSupportedEventTypes (*C++ function*), 247
nvmlDeviceGetSupportedGraphicsClocks (*C++ function*), 195
nvmlDeviceGetSupportedMemoryClocks (*C++ function*), 196
nvmlDeviceGetSupportedPerformanceStates (*C++ function*), 197
nvmlDeviceGetSupportedVgpus (*C++ function*), 454
nvmlDeviceGetTargetFanSpeed (*C++ function*), 198
nvmlDeviceGetTemperature (*C++ function*), 198
nvmlDeviceGetTemperatureThreshold (*C++ function*), 198
nvmlDeviceGetTemperatureV (*C++ function*), 199
nvmlDeviceGetThermalSettings (*C++ function*), 200
nvmlDeviceGetTopologyCommonAncestor (*C++ function*), 200
nvmlDeviceGetTopologyNearestGpus (*C++ function*), 201
nvmlDeviceGetTotalEccErrors (*C++ function*), 201
nvmlDeviceGetTotalEnergyConsumption (*C++ function*), 202
nvmlDeviceGetUnrepairableMemoryFlag_v1 (*C++ function*), 100
nvmlDeviceGetUtilizationRates (*C++ function*), 203
nvmlDeviceGetUUID (*C++ function*), 202
nvmlDeviceGetVbiosVersion (*C++ function*), 204
nvmlDeviceGetVgpuCapabilities (*C++ function*), 455
nvmlDeviceGetVgpuHeterogeneousMode (*C++ function*), 428
nvmlDeviceGetVgpuInstancesUtilizationInfo (*C++ function*), 489
nvmlDeviceGetVgpuMetadata (*C++ function*), 483
nvmlDeviceGetVgpuProcessesUtilizationInfo (*C++ function*), 491
nvmlDeviceGetVgpuProcessUtilization (*C++ function*), 490
nvmlDeviceGetVgpuSchedulerCapabilities (*C++ function*), 483
nvmlDeviceGetVgpuSchedulerLog (*C++ function*), 484
nvmlDeviceGetVgpuSchedulerLog_v2 (*C++ function*), 455
nvmlDeviceGetVgpuSchedulerState (*C++ function*), 484
nvmlDeviceGetVgpuSchedulerState_v2 (*C++ function*), 456
nvmlDeviceGetVgpuTypeCreatablePlacements (*C++ function*), 428
nvmlDeviceGetVgpuTypeSupportedPlacements (*C++ function*), 429
nvmlDeviceGetVgpuUtilization (*C++ function*), 492
nvmlDeviceGetViolationStatus (*C++ function*), 205
nvmlDeviceGetVirtualizationMode (*C++ function*), 430
nvmlDeviceGpuRecoveryAction_t (*C++ enum*), 448
nvmlDeviceGpuRecoveryAction_t::NVML_GPU_RECOVERY_AC (*C++ enumerator*), 448
nvmlDeviceGpuRecoveryAction_t::NVML_GPU_RECOVERY_AC (*C++ enumerator*), 448
nvmlDeviceGpuRecoveryAction_t::NVML_GPU_RECOVERY_AC (*C++ enumerator*), 448
nvmlDeviceGpuRecoveryAction_t::NVML_GPU_RECOVERY_AC (*C++ enumerator*), 448
nvmlDeviceGpuRecoveryAction_t::NVML_GPU_RECOVERY_AC (*C++ enumerator*), 448
nvmlDeviceGpuRecoveryAction_t::NVML_GPU_RECOVERY_AC (*C++ enumerator*), 448
nvmlDeviceGpuRecoveryAction_t::NVML_GPU_RECOVERY_AC (*C++ enumerator*), 448
nvmlDeviceIsMigDeviceHandle (*C++ function*), 331
nvmlDeviceLockNvlinkRemapTable_v1 (*C++ function*), 389
nvmlDeviceModifyDrainState (*C++ function*),

235
nvmlDeviceOnSameBoard (*C++ function*), 205
nvmlDevicePerfMetricsGetSamples_v1 (*C++ function*), 205
nvmlDevicePerfModes_t (*C++ type*), 94
nvmlDevicePerfModes_v1 (*C macro*), 82
nvmlDevicePerfModes_v1_t (*C++ struct*), 516
nvmlDevicePerfModes_v1_t::str (*C++ member*), 516
nvmlDevicePerfModes_v1_t::version (*C++ member*), 516
nvmlDevicePowerMizerModes_v1_t (*C++ struct*), 516
nvmlDevicePowerMizerModes_v1_t::currentMode (*C++ member*), 516
nvmlDevicePowerMizerModes_v1_t::mode (*C++ member*), 516
nvmlDevicePowerMizerModes_v1_t::supportedPowerMizerModes (*C++ member*), 516
nvmlDevicePowerSmoothingActivatePresetProfile (*C++ function*), 412
nvmlDevicePowerSmoothingSetState (*C++ function*), 413
nvmlDevicePowerSmoothingUpdatePresetProfileParam (*C++ function*), 413
nvmlDeviceQueryDrainState (*C++ function*), 236
nvmlDeviceReadPRMCounters_v1 (*C++ function*), 402
nvmlDeviceReadWritePRM_v1 (*C++ function*), 402
nvmlDeviceRegisterEvents (*C++ function*), 248
nvmlDeviceRemoveGpu_v2 (*C++ function*), 236
nvmlDeviceResetApplicationsClocks (*C++ function*), 56
nvmlDeviceResetGpuLockedClocks (*C++ function*), 56
nvmlDeviceResetMemoryLockedClocks (*C++ function*), 57
nvmlDeviceResetNvLinkErrorCounters (*C++ function*), 389
nvmlDeviceResetNvLinkUtilizationCounter (*C++ function*), 389
nvmlDeviceSetAccountingMode (*C++ function*), 44
nvmlDeviceSetAdaptiveTgpMode_v1 (*C++ function*), 206
nvmlDeviceSetAPIRestriction (*C++ function*), 57
nvmlDeviceSetApplicationsClocks (*C++ function*), 58
nvmlDeviceSetAutoBoostedClocksEnabled (*C++ function*), 58
nvmlDeviceSetClockOffsets (*C++ function*), 207
nvmlDeviceSetComputeMode (*C++ function*), 59
nvmlDeviceSetConfComputeUnprotectedMemSize (*C++ function*), 207
nvmlDeviceSetCpuAffinity (*C++ function*), 101
nvmlDeviceSetDefaultAutoBoostedClocksEnabled (*C++ function*), 60
nvmlDeviceSetDefaultFanSpeed_v2 (*C++ function*), 61
nvmlDeviceSetDramEncryptionMode (*C++ function*), 207
nvmlDeviceSetDriverModel (*C++ function*), 61
nvmlDeviceSetEccMode (*C++ function*), 62
nvmlDeviceSetFanControlPolicy (*C++ function*), 63
nvmlDeviceSetFanSpeed_v2 (*C++ function*), 63
nvmlDeviceSetGpcClkVfOffset (*C++ function*), 64
nvmlDeviceSetGpuLockedClocks (*C++ function*), 64
nvmlDeviceSetGpuOperationMode (*C++ function*), 65
nvmlDeviceSetHostname_v1 (*C++ function*), 208
nvmlDeviceSetMemClkVfOffset (*C++ function*), 66
nvmlDeviceSetMemoryLimits_v1 (*C++ function*), 209
nvmlDeviceSetMemoryLockedClocks (*C++ function*), 67
nvmlDeviceSetMigMode (*C++ function*), 332
nvmlDeviceSetNvlinkBwMode (*C++ function*), 391
nvmlDeviceSetNvlinkBwModeAsync_v1 (*C++ function*), 391
nvmlDeviceSetNvLinkDeviceLowPowerThreshold (*C++ function*), 390
nvmlDeviceSetNvlinkNvleReady_v1 (*C++ function*), 392
nvmlDeviceSetNvLinkUtilizationControl (*C++ function*), 390
nvmlDeviceSetPersistenceMode (*C++ function*), 67
nvmlDeviceSetPowerManagementLimit (*C++ function*), 68
nvmlDeviceSetPowerManagementLimit_v2 (*C++ function*), 210
nvmlDeviceSetPowerMizerMode_v1 (*C++ function*), 211
nvmlDeviceSetRusdSettings_v1 (*C++ function*), 69
nvmlDeviceSetTemperatureThreshold (*C++*

function), 70
nvmlDeviceSetupNvlinkNvleEncryptionKey_v1 (C++ function), 392
nvmlDeviceSetVgpuCapabilities (C++ function), 430
nvmlDeviceSetVgpuHeterogeneousMode (C++ function), 431
nvmlDeviceSetVgpuSchedulerState (C++ function), 485
nvmlDeviceSetVgpuSchedulerState_v2 (C++ function), 456
nvmlDeviceSetVirtualizationMode (C++ function), 432
nvmlDeviceValidateInforom (C++ function), 211
nvmlDeviceVgpuCapability_t (C++ enum), 438
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_COMPUTE, 438
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_MAX, 439
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_MIG_TIMESLICING (C++ enumerator), 438
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_MIG_TIMESLICING_ENABLED (C++ enumerator), 438
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_MIG_TIMESLICING_SUPPORTED (C++ enumerator), 438
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_MINI_QUARTER_GPU (C++ enumerator), 438
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_READ_DEVICE_BUFFER_BW (C++ enumerator), 438
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_UVM_STUN (C++ enumerator), 439
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_WARM_UPDATE (C++ enumerator), 438
nvmlDeviceVgpuCapability_t::NVML_DEVICE_VGPU_CAP_WRITE_DEVICE_BUFFER_BW (C++ enumerator), 438
nvmlDeviceVgpuForceGspUnload (C++ function), 432
nvmlDeviceWorkloadPowerProfileClearRequestedProfiles (C++ function), 407
nvmlDeviceWorkloadPowerProfileGetCurrentProfiles (C++ function), 407
nvmlDeviceWorkloadPowerProfileGetProfilesInfo (C++ function), 408
nvmlDeviceWorkloadPowerProfileSetRequestedProfiles (C++ function), 408
nvmlDeviceWorkloadPowerProfileUpdateProfiles_v1 (C++ function), 409
nvmlDramEncryptionInfo_t (C++ type), 94
nvmlDramEncryptionInfo_v1 (C macro), 82
nvmlDramEncryptionInfo_v1_t (C++ struct), 517
nvmlDramEncryptionInfo_v1_t::encryptionState (C++ member), 517
nvmlDramEncryptionInfo_v1_t::version (C++ member), 517
nvmlDriverModel_t (C++ enum), 86
nvmlDriverModel_t::NVML_DRIVER_MCDM (C++ enumerator), 86
nvmlDriverModel_t::NVML_DRIVER_WDDM (C++ enumerator), 86
nvmlDriverModel_t::NVML_DRIVER_WDM (C++ enumerator), 86
nvmlEccBankRemapperHistogram_v1_t (C++ struct), 517
nvmlEccBankRemapperHistogram_v1_t::maxSpareGroupCount (C++ member), 517
nvmlEccBankRemapperHistogram_v1_t::noSpareGroupCount (C++ member), 517
nvmlEccBankRemapperStatus_v1_t (C++ struct), 517
nvmlEccBankRemapperStatus_v1_t::activeRemappings (C++ member), 517
nvmlEccBankRemapperStatus_v1_t::bPending (C++ member), 518
nvmlEccBankRemapperStatus_v1_t::histogram (C++ member), 518
nvmlEccBankRemapperStatus_v1_t::inactiveRemappings (C++ member), 517
nvmlEccBitType_t (C macro), 82
nvmlEccCounterType_t (C++ enum), 86
nvmlEccCounterType_t::NVML_AGGREGATE_ECC (C++ enumerator), 86
nvmlEccCounterType_t::NVML_ECC_COUNTER_TYPE_COUNT (C++ enumerator), 86
nvmlEccCounterType_t::NVML_VOLATILE_ECC (C++ enumerator), 86
nvmlEccErrorCounts_t (C++ struct), 518
nvmlEccErrorCounts_t::deviceMemory (C++ member), 518
nvmlEccErrorCounts_t::l1Cache (C++ member), 518
nvmlEccErrorCounts_t::l2Cache (C++ member), 518
nvmlEccErrorCounts_t::registerFile (C++ member), 518
nvmlEccSramErrorStatus_t (C++ type), 94
nvmlEccSramErrorStatus_v1 (C macro), 82
nvmlEccSramErrorStatus_v1_t (C++ struct),

518
nvmlEccSramErrorStatus_v1_t::aggregateCor (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::aggregateUncBucketL2 (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::aggregateUncBucketMcu (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::aggregateUncBucketOther (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::aggregateUncBucketPcie (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::aggregateUncBucketSm (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::aggregateUncParity (*C++ member*), 518
nvmlEccSramErrorStatus_v1_t::aggregateUncSecDed (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::bThresholdExceeded (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::version (*C++ member*), 518
nvmlEccSramErrorStatus_v1_t::volatileCor (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::volatileUncParity (*C++ member*), 519
nvmlEccSramErrorStatus_v1_t::volatileUncSecDed (*C++ member*), 519
nvmlEccSramUniqueUncorrectedErrorCounts_t (*C++ type*), 94
nvmlEccSramUniqueUncorrectedErrorCounts_v1 (*C macro*), 83
nvmlEccSramUniqueUncorrectedErrorCounts_v1_t (*C++ struct*), 519
nvmlEccSramUniqueUncorrectedErrorCounts_v1_t::entries (*C++ member*), 520
nvmlEccSramUniqueUncorrectedErrorCounts_v1_t::numEntries (*C++ member*), 519
nvmlEccSramUniqueUncorrectedErrorCounts_v1_t::version (*C++ member*), 519
nvmlEccSramUniqueUncorrectedErrorEntry_v1 (*C++ struct*), 520
nvmlEccSramUniqueUncorrectedErrorEntry_v1_t::address (*C++ member*), 520
nvmlEccSramUniqueUncorrectedErrorEntry_v1_t::count (*C++ member*), 520
nvmlEccSramUniqueUncorrectedErrorEntry_v1_t::isParity (*C++ member*), 520
nvmlEccSramUniqueUncorrectedErrorEntry_v1_t::unitId (*C++ member*), 520
nvmlEnableState_t (*C++ enum*), 87
nvmlEnableState_t::NVML_FEATURE_DISABLED (*C++ enumerator*), 87
nvmlEnableState_t::NVML_FEATURE_ENABLED (*C++ enumerator*), 87
nvmlEncoderSessionInfo_t (*C++ struct*), 520
nvmlEncoderSessionInfo_t::averageFps (*C++ member*), 521
nvmlEncoderSessionInfo_t::averageLatency (*C++ member*), 521
nvmlEncoderSessionInfo_t::codecType (*C++ member*), 521
nvmlEncoderSessionInfo_t::hResolution (*C++ member*), 521
nvmlEncoderSessionInfo_t::pid (*C++ member*), 520
nvmlEncoderSessionInfo_t::sessionId (*C++ member*), 520
nvmlEncoderSessionInfo_t::vgpuInstance (*C++ member*), 521
nvmlEncoderSessionInfo_t::vResolution (*C++ member*), 521
nvmlEncoderType_t (*C++ enum*), 237
nvmlEncoderType_t::NVML_ENCODER_QUERY_AV1 (*C++ enumerator*), 237
nvmlEncoderType_t::NVML_ENCODER_QUERY_H264 (*C++ enumerator*), 237
nvmlEncoderType_t::NVML_ENCODER_QUERY_HEVC (*C++ enumerator*), 237
nvmlEncoderType_t::NVML_ENCODER_QUERY_UNKNOWN (*C++ enumerator*), 237
nvmlErrorString (*C++ function*), 238
nvmlEventData_t (*C++ struct*), 521
nvmlEventData_t::computeInstanceId (*C++ member*), 521
nvmlEventData_t::device (*C++ member*), 521
nvmlEventData_t::eventData (*C++ member*), 521
nvmlEventData_t::eventType (*C++ member*), 521
nvmlEventData_t::gpuInstanceId (*C++ member*), 521
nvmlEventDataType_t (*C++ enum*), 245
nvmlEventDataType_t::NVML_EVENT_DATA_TYPE_GPU_OPERA (*C++ enumerator*), 245
nvmlEventDataType_t::NVML_EVENT_DATA_TYPE_NVML_EVEN (*C++ enumerator*), 245
nvmlEventMigConfigChange (*C macro*), 239
nvmlEventSet_t (*C++ type*), 260
nvmlEventSetCreate (*C++ function*), 249
nvmlEventSetFree (*C++ function*), 249
nvmlEventSetGetContextCount_v1 (*C++ function*), 250

nvmlEventSetGetContextCount_v1_t (*C++ struct*), 522
nvmlEventSetGetContextCount_v1_t::count (*C++ member*), 522
nvmlEventSetGetContextData_v1 (*C++ function*), 251
nvmlEventSetGetContextData_v1_t (*C++ struct*), 522
nvmlEventSetGetContextData_v1_t::data (*C++ member*), 522
nvmlEventSetGetContextData_v1_t::dataSize (*C++ member*), 522
nvmlEventSetGetContextData_v1_t::index (*C++ member*), 522
nvmlEventSetGetContextInfo_v1 (*C++ function*), 252
nvmlEventSetGetContextInfo_v1_t (*C++ struct*), 522
nvmlEventSetGetContextInfo_v1_t::dataFormatVersion (*C++ member*), 522
nvmlEventSetGetContextInfo_v1_t::dataSize (*C++ member*), 522
nvmlEventSetGetContextInfo_v1_t::index (*C++ member*), 522
nvmlEventSetGetContextInfo_v1_t::nvmlGpuOperationalEventContextType (*C++ member*), 522
nvmlEventSetGetContextInfo_v1_t::sourceEventComponentType (*C++ member*), 522
nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1 (*C++ function*), 253
nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t (*C++ struct*), 523
nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t::index (*C++ member*), 523
nvmlEventSetGetGpuOperationalEventContextLegacyXid_v1_t::xidCode (*C++ member*), 523
nvmlEventSetRegisterGpuOperationalEvents_v1 (*C++ function*), 254
nvmlEventSetWait_v2 (*C++ function*), 255
nvmlEventSetWait_v3 (*C++ function*), 256
nvmlEventSetWait_v3_t (*C++ struct*), 523
nvmlEventSetWait_v3_t::attributes (*C++ member*), 525
nvmlEventSetWait_v3_t::categoryId (*C++ member*), 524
nvmlEventSetWait_v3_t::chipletId (*C++ member*), 525
nvmlEventSetWait_v3_t::computeInstanceId (*C++ member*), 524
nvmlEventSetWait_v3_t::dataType (*C++ member*), 523
nvmlEventSetWait_v3_t::eventData (*C++ member*), 524
nvmlEventSetWait_v3_t::eventType (*C++ member*), 523
nvmlEventSetWait_v3_t::gpuInstanceId (*C++ member*), 524
nvmlEventSetWait_v3_t::groupAttributes (*C++ member*), 525
nvmlEventSetWait_v3_t::groupCperSize (*C++ member*), 525
nvmlEventSetWait_v3_t::groupCursor (*C++ member*), 524
nvmlEventSetWait_v3_t::groupIndex (*C++ member*), 525
nvmlEventSetWait_v3_t::groupSize (*C++ member*), 525
nvmlEventSetWait_v3_t::instanceId (*C++ member*), 524
nvmlEventSetWait_v3_t::logLevel (*C++ member*), 525
nvmlEventSetWait_v3_t::moduleEventCode (*C++ member*), 524
nvmlEventSetWait_v3_t::moduleInstance (*C++ member*), 524
nvmlEventSetWait_v3_t::originator (*C++ member*), 524
nvmlEventSetWait_v3_t::scope (*C++ member*), 524
nvmlEventSetWait_v3_t::severity (*C++ member*), 524
nvmlEventSetWait_v3_t::sourceModule (*C++ member*), 523
nvmlEventSetWait_v3_t::timeoutMs (*C++ member*), 523
nvmlEventSetWait_v3_t::timestampUsec (*C++ member*), 524
nvmlEventSetWait_v3_t::traceId (*C++ member*), 524
nvmlEventSetWait_v3_t::uuid (*C++ member*), 523
nvmlEventTypeAll (*C macro*), 239
nvmlEventTypeClock (*C macro*), 239
nvmlEventTypeDoubleBitEccError (*C macro*), 239
nvmlEventTypeDramRetirementEvent (*C macro*), 240
nvmlEventTypeDramRetirementFailure (*C macro*), 240
nvmlEventTypeFatalPoisonError (*C macro*), 240
nvmlEventTypeGpuRecoveryAction (*C macro*), 240
nvmlEventTypeGpuUnavailableError (*C macro*), 240
nvmlEventTypeNone (*C macro*), 240
nvmlEventTypeNonFatalPoisonError (*C macro*), 240

nvmlEventTypePowerSourceChange (*C macro*), 240
nvmlEventTypePState (*C macro*), 240
nvmlEventTypeSingleBitEccError (*C macro*), 241
nvmlEventTypeSingleBitEccErrorStorm (*C macro*), 241
nvmlEventTypeXidCriticalError (*C macro*), 241
nvmlExcludedDeviceInfo_t (*C++ struct*), 525
nvmlExcludedDeviceInfo_t::pciInfo (*C++ member*), 525
nvmlExcludedDeviceInfo_t::uuid (*C++ member*), 525
nvmlFanControlPolicy_t (*C++ type*), 94
nvmlFanSpeedInfo_t (*C++ type*), 94
nvmlFanSpeedInfo_v1 (*C macro*), 83
nvmlFanSpeedInfo_v1_t (*C++ struct*), 527
nvmlFanSpeedInfo_v1_t::fan (*C++ member*), 527
nvmlFanSpeedInfo_v1_t::speed (*C++ member*), 527
nvmlFanSpeedInfo_v1_t::version (*C++ member*), 527
nvmlFanState_t (*C++ enum*), 425
nvmlFanState_t::NVML_FAN_FAILED (*C++ enumerator*), 426
nvmlFanState_t::NVML_FAN_NORMAL (*C++ enumerator*), 425
nvmlFBCSessionInfo_t (*C++ struct*), 525
nvmlFBCSessionInfo_t::averageFPS (*C++ member*), 526
nvmlFBCSessionInfo_t::averageLatency (*C++ member*), 526
nvmlFBCSessionInfo_t::displayOrdinal (*C++ member*), 526
nvmlFBCSessionInfo_t::hMaxResolution (*C++ member*), 526
nvmlFBCSessionInfo_t::hResolution (*C++ member*), 526
nvmlFBCSessionInfo_t::pid (*C++ member*), 526
nvmlFBCSessionInfo_t::sessionFlags (*C++ member*), 526
nvmlFBCSessionInfo_t::sessionId (*C++ member*), 526
nvmlFBCSessionInfo_t::sessionType (*C++ member*), 526
nvmlFBCSessionInfo_t::vgpuInstance (*C++ member*), 526
nvmlFBCSessionInfo_t::vMaxResolution (*C++ member*), 526
nvmlFBCSessionInfo_t::vResolution (*C++ member*), 526
nvmlFBCSessionType_t (*C++ enum*), 310
nvmlFBCSessionType_t::NVML_FBC_SESSION_TYPE_CUDA (*C++ enumerator*), 310
nvmlFBCSessionType_t::NVML_FBC_SESSION_TYPE_HWENC (*C++ enumerator*), 311
nvmlFBCSessionType_t::NVML_FBC_SESSION_TYPE_TOSYS (*C++ enumerator*), 310
nvmlFBCSessionType_t::NVML_FBC_SESSION_TYPE_UNKNOWN (*C++ enumerator*), 310
nvmlFBCSessionType_t::NVML_FBC_SESSION_TYPE_VID (*C++ enumerator*), 311
nvmlFBCStats_t (*C++ struct*), 526
nvmlFBCStats_t::averageFPS (*C++ member*), 527
nvmlFBCStats_t::averageLatency (*C++ member*), 527
nvmlFBCStats_t::sessionsCount (*C++ member*), 527
nvmlFieldValue_t (*C++ struct*), 527
nvmlFieldValue_t::fieldId (*C++ member*), 527
nvmlFieldValue_t::latencyUsec (*C++ member*), 527
nvmlFieldValue_t::nvmlReturn (*C++ member*), 528
nvmlFieldValue_t::scopeId (*C++ member*), 527
nvmlFieldValue_t::timestamp (*C++ member*), 527
nvmlFieldValue_t::value (*C++ member*), 528
nvmlFieldValue_t::valueType (*C++ member*), 528
nvmlFlagDefault (*C macro*), 83
nvmlFlagForce (*C macro*), 83
nvmlGetCPER_v1_t (*C++ struct*), 528
nvmlGetCPER_v1_t::buffer (*C++ member*), 528
nvmlGetCPER_v1_t::bufferSize (*C++ member*), 528
nvmlGetCPER_v1_t::cursor (*C++ member*), 528
nvmlGetExcludedDeviceCount (*C++ function*), 260
nvmlGetExcludedDeviceInfoByIndex (*C++ function*), 260
nvmlGetMemoryLimits_v1_t (*C++ struct*), 528
nvmlGetMemoryLimits_v1_t::currentUsed (*C++ member*), 529
nvmlGetMemoryLimits_v1_t::hardLimit (*C++ member*), 529
nvmlGetMemoryLimits_v1_t::nameSpace (*C++ member*), 529
nvmlGetMemoryLimits_v1_t::softLimit (*C++ member*), 529

nvmlGetVgpuCompatibility (*C++ function*), 485
nvmlGetVgpuDriverCapabilities (*C++ function*), 457
nvmlGetVgpuVersion (*C++ function*), 486
nvmlGpmMetric_t (*C++ struct*), 529
nvmlGpmMetric_t::longName (*C++ member*), 529
nvmlGpmMetric_t::metricId (*C++ member*), 529
nvmlGpmMetric_t::metricInfo (*C++ member*), 529
nvmlGpmMetric_t::nvmlReturn (*C++ member*), 529
nvmlGpmMetric_t::shortName (*C++ member*), 529
nvmlGpmMetric_t::unit (*C++ member*), 529
nvmlGpmMetric_t::value (*C++ member*), 529
nvmlGpmMetricId_t (*C++ enum*), 339
nvmlGpmMetricId_t::NVML_GPM_METRIC_ANY_TENSOR_UTIL (*C++ enumerator*), 339
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_DATA_RX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_DATA_TX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK0_DATA_RX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK0_DATA_TX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK0_TOTAL_RX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK0_TOTAL_TX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK0_TOTAL_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK1_DATA_RX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK1_DATA_TX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK1_TOTAL_RX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK1_TOTAL_TX_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK1_TOTAL_PER_SEC (*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK2_DATA_RX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK2_DATA_TX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK2_TOTAL_RX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK2_TOTAL_TX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK2_TOTAL_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK3_DATA_RX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK3_DATA_TX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK3_TOTAL_RX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK3_TOTAL_TX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK3_TOTAL_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK4_DATA_RX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK4_DATA_TX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK4_TOTAL_RX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK4_TOTAL_TX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK4_TOTAL_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK5_DATA_RX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK5_DATA_TX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK5_TOTAL_RX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK5_TOTAL_TX_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK5_TOTAL_PER_SEC (*C++ enumerator*), 345
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK6_DATA_RX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK6_DATA_TX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK6_TOTAL_RX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK6_TOTAL_TX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK6_TOTAL_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK7_DATA_RX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK7_DATA_TX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK7_TOTAL_RX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK7_TOTAL_TX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK7_TOTAL_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK8_DATA_RX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK8_DATA_TX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK8_TOTAL_RX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK8_TOTAL_TX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK8_TOTAL_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK9_DATA_RX_PER_SEC (*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK9_DATA_TX_PER_SEC (*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK9_TOTAL_RX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK9_TOTAL_TX_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK9_TOTAL_PER_SEC (*C++ enumerator*), 346
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK10_DATA_RX_PER_SEC (*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK10_DATA_TX_PER_SEC

(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_TOTAL_RX_PER_SEC
(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_TOTAL_TX_PER_SEC
(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_0_DATA_RX_PER_SEC
(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_0_DATA_TX_PER_SEC
(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_0_RAW_RX_PER_SEC
(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_0_RAW_TX_PER_SEC
(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_1_DATA_RX_PER_SEC
(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_1_DATA_TX_PER_SEC
(*C++ enumerator*), 347
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_1_RAW_RX_PER_SEC
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_LINK_1_RAW_TX_PER_SEC
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_TOTAL_DATA_RX_PER_SEC
(*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_C2C_TOTAL_DATA_TX_PER_SEC
(*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_DFMA_TENSOR_UTIL
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_DMMA_TENSOR_UTIL
(*C++ enumerator*), 339
nvmlGpmMetricId_t::NVML_GPM_METRIC_DRAM_BW_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_DRAM_CYCLES_ACTIVE
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_FP16_CYCLES_ACTIVE
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_FP16_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_FP32_CYCLES_ACTIVE
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_FP32_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_FP64_CYCLES_ACTIVE
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_FP64_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR0_CTXSW_ACTIVE
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR0_CTXSW_CYCLES_ELAPSED
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR0_CTXSW_CYCLES_IN_USE
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR0_CTXSW_REQUESTS
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR1_CTXSW_ACTIVE
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR1_CTXSW_CYCLES_ELAPSED
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR1_CTXSW_CYCLES_IN_USE
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR1_CTXSW_REQUESTS
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR2_CTXSW_ACTIVE
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR2_CTXSW_CYCLES_ELAPSED
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR2_CTXSW_CYCLES_IN_USE
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR2_CTXSW_REQUESTS
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR3_CTXSW_ACTIVE
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR3_CTXSW_CYCLES_ELAPSED
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR3_CTXSW_CYCLES_IN_USE
(*C++ enumerator*), 349
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR3_CTXSW_REQUESTS
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR4_CTXSW_ACTIVE
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR4_CTXSW_CYCLES_ELAPSED
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR4_CTXSW_CYCLES_IN_USE
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR4_CTXSW_REQUESTS

(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR5_CYCLES_ACTIVE
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR5_CYCLES_ELAPSED
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR5_CYCLES_PER_REQ
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR5_SM_UTIL
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR6_CYCLES_ACTIVE
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR6_CYCLES_ELAPSED
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR6_CYCLES_PER_REQ
(*C++ enumerator*), 350
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR6_SM_UTIL
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR7_CYCLES_ACTIVE
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR7_CYCLES_ELAPSED
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR7_CYCLES_PER_REQ
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_GR7_SM_UTIL
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_GRAPHICS_UTIL
(*C++ enumerator*), 339
nvmlGpmMetricId_t::NVML_GPM_METRIC_HMMA_CYCLES_ACTIVE
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_HMMA_TENSOR_UTIL
(*C++ enumerator*), 339
nvmlGpmMetricId_t::NVML_GPM_METRIC_HOSTMEM_CACHE_HIT
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_HOSTMEM_CACHE_MISS
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_IMMA_CYCLES_ACTIVE
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_IMMA_TENSOR_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_INTEGER_CYCLES_ACTIVE
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_INTEGER_UTIL
(*C++ enumerator*), 339
nvmlGpmMetricId_t::NVML_GPM_METRIC_MAX
(*C++ enumerator*), 369
nvmlGpmMetricId_t::NVML_GPM_METRIC_MMA_CYCLES_ACTIVE
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVDEC_0_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVDEC_1_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVDEC_2_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVDEC_3_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVDEC_4_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVDEC_5_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVDEC_6_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVDEC_7_UTIL
(*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVENC_0_UTIL
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVENC_1_UTIL
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVENC_2_UTIL
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVENC_3_UTIL
(*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVJPG_0_UTIL
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVJPG_1_UTIL
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVJPG_2_UTIL
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVJPG_3_UTIL
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVJPG_4_UTIL
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVJPG_5_UTIL
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVJPG_6_UTIL
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVJPG_7_UTIL
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L0_RX
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L0_RX_PER
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L0_TX
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L0_TX_PER
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L1_RX
(*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L1_RX_PER
(*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L1_TX
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L1_TX_PER

(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L2_RX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L2_TOTAL_RX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L2_TOTAL_TX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L2_TX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L3_RX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L3_TOTAL_RX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L3_TOTAL_TX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L3_TX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L4_RX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L4_TOTAL_RX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L4_TOTAL_TX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L4_TX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L5_RX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L5_TOTAL_RX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L5_TOTAL_TX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L5_TX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L6_RX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L6_TOTAL_RX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L6_TOTAL_TX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L6_TX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L7_RX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L7_TOTAL_RX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L7_TOTAL_TX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L7_TX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L8_RX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L8_TOTAL_RX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L8_TOTAL_TX_PER_SEC
(*C++ enumerator*), 355
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L8_TX_PER_SEC
(*C++ enumerator*), 342
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L9_RX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L9_RX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L9_TX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L9_TX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L10_RX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L10_RX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L10_TX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L10_TX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L11_RX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L11_RX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L11_TX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L11_TX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L12_RX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L12_RX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L12_TX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L12_TX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L13_RX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L13_RX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L13_TX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L13_TX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L14_RX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L14_RX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L14_TX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L14_TX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L15_RX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L15_RX_PER_SEC

(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L16_RX_PER_SEC
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L16_TX_PER_SEC
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L16_RX_PKTS
(*C++ enumerator*), 356
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L16_TX_PKTS
(*C++ enumerator*), 343
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L16_RX_PER_SEC
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L16_TX_PER_SEC
(*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L17_RX_PKTS
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L17_TX_PKTS
(*C++ enumerator*), 344
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L17_RX_PER_SEC
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L17_TX_PER_SEC
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L18_RX_PKTS
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L18_TX_PKTS
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L18_RX_PER_SEC
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L18_TX_PER_SEC
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L19_RX_PKTS
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L19_TX_PKTS
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L19_RX_PER_SEC
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L19_TX_PER_SEC
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L20_RX_PKTS
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L20_TX_PKTS
(*C++ enumerator*), 351
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L20_RX_PER_SEC
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L20_TX_PER_SEC
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L21_RX_PKTS
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L21_TX_PKTS
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L21_RX_PER_SEC
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L21_TX_PER_SEC
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L22_RX_PKTS
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L22_TX_PKTS
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L22_RX_PER_SEC
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L22_TX_PER_SEC
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L23_RX_PKTS
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L23_TX_PKTS
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L23_RX_PER_SEC
(*C++ enumerator*), 357
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L23_TX_PER_SEC
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L24_RX_PKTS
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L24_TX_PKTS
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L24_RX_PER_SEC
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L24_TX_PER_SEC
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L25_RX_PKTS
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L25_TX_PKTS
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L25_RX_PER_SEC
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L25_TX_PER_SEC
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L26_RX_PKTS
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L26_TX_PKTS
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L26_RX_PER_SEC
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L26_TX_PER_SEC
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L27_RX_PKTS
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L27_TX_PKTS
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L27_RX_PER_SEC
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L27_TX_PER_SEC
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L28_RX_PKTS
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L28_TX_PKTS
(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L28_RX_PER_SEC
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L28_TX_PER_SEC
(*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L29_RX_PKTS

(*C++ enumerator*), 352
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L29_RX_PER_SEC (*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L29_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L29_TX_PER_SEC (*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L30_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L30_RX_PER_SEC (*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L30_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L30_TX_PER_SEC (*C++ enumerator*), 358
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L31_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L31_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L31_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L31_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L32_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L32_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L32_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L32_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L33_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L33_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L33_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L33_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L34_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L34_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L34_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L34_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L35_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L35_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L35_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L35_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L36_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L36_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L36_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L36_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L37_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L37_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L37_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L37_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L38_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L38_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L38_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L38_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L39_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L39_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L39_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L39_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L40_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L40_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L40_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L40_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L41_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L41_RX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L41_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L41_TX_PER_SEC (*C++ enumerator*), 359
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L42_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L42_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L42_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L42_TX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L43_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L43_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L43_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L43_TX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L44_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L44_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L44_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L44_TX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L45_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L45_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L45_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L45_TX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L46_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L46_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L46_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L46_TX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L47_RX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L47_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L47_TX (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L47_TX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_TOTAL_RX_PER_SEC (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_TOTAL_TX_PER_SEC (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_PCIE_RX_PER_SEC (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_PCIE_TX_PER_SEC (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_SM_OCCUPANCY (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_SYNC_BOOST (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_TENSOR_ACTIVE (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_VIDDEC_UTIL (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_VIDENC_UTIL (*C++ enumerator*), 353
nvmlGpmSample_t (*C++ type*), 353
nvmlGpmSampleAlloc (*C++ function*), 353
nvmlGpmSampleFree (*C++ function*), 354
nvmlGpmSampleGet (*C++ function*), 354
nvmlGpmSetStreamingEnabled (*C++ function*), 355
nvmlGpmSupport_t (*C++ type*), 355
nvmlGpmSupport_t::gpmSupportedDevices (*C++ member*), 355
nvmlGpmSupport_t::version (*C++ member*), 355
nvmlGpmSupportGet (*C++ function*), 355
nvmlGpuDynamicPstatesInfo_t (*C++ type*), 355
nvmlGpuDynamicPstatesInfo_t::flags (*C++ member*), 355
nvmlGpuDynamicPstatesInfo_t::utilization (*C++ member*), 355
nvmlGpuDynamicPstatesInfo_t::utilization::bIsPresent (*C++ member*), 355
nvmlGpuDynamicPstatesInfo_t::utilization::percentage (*C++ member*), 355
nvmlGpuDynamicPstatesInfo_t::utilization::reserved (*C++ member*), 355
nvmlGpuDynamicPstatesInfo_t::version (*C++ member*), 355
nvmlGpuFabricInfo_t (*C++ type*), 355
nvmlGpuFabricInfo_t::clusterUuid (*C++ member*), 355
nvmlGpuFabricInfo_t::cliqueId (*C++ member*), 355
nvmlGpuFabricInfo_t::healthMask (*C++ member*), 355
nvmlGpuFabricInfo_t::healthSummary (*C++ member*), 355
nvmlGpuFabricInfo_t::state (*C++ member*), 355
nvmlGpuFabricInfo_t::status (*C++ member*), 355
nvmlGpuFabricInfo_t::version (*C++ member*), 355
nvmlGpuFabricInfoV (*C++ function*), 355
nvmlGpuFabricInfoV_t (*C++ type*), 355
nvmlGpuFabricInfoV_t::clusterUuid (*C++ member*), 355
nvmlGpuFabricInfoV_t::cliqueId (*C++ member*), 355
nvmlGpuFabricInfoV_t::healthMask (*C++ member*), 355
nvmlGpuFabricInfoV_t::healthSummary (*C++ member*), 355
nvmlGpuFabricInfoV_t::state (*C++ member*), 355
nvmlGpuFabricInfoV_t::status (*C++ member*), 355
nvmlGpuFabricInfoV_t::version (*C++ member*), 355
nvmlGpuInstanceCreateComputeInstance (*C++ function*), 355
nvmlGpuInstanceCreateComputeInstanceWithProfileInfo (*C++ function*), 356
nvmlGpuInstanceDestroy (*C++ function*), 356
nvmlGpuInstanceGetComputeInstanceById (*C++ function*), 356
nvmlGpuInstanceGetComputeInstanceProfileInfo (*C++ function*), 356
nvmlGpuInstanceGetComputeInstanceProfileInfoV (*C++ function*), 356
nvmlGpuInstanceGetComputeInstanceRemainingCapacity (*C++ function*), 356
nvmlGpuInstanceGetComputeInstances (*C++ function*), 356
nvmlGpuInstanceGetInfo (*C++ function*), 356
nvmlGpuInstanceGetMemoryAffinityMask (*C++ function*), 356
nvmlGpuInstanceGetMemoryInfo (*C++ function*), 356
nvmlGpuInstanceGetMemoryInfo_v2 (*C++ function*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t (*C++ type*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::bar1Memory (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::bar1Total (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::bar1Used (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::bar1Free (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::version (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::memory (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::total (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::used (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::free (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::version (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::reserved (*C++ member*), 356
nvmlGpuInstanceGetMemoryInfo_v2_t::version (*C++ member*), 356
nvmlGpuInstanceGetMigDeviceHandleByIndex (*C++ function*), 356
nvmlGpuInstanceGetPlacements (*C++ function*), 356
nvmlGpuInstanceGetProfileInfo (*C++ function*), 356
nvmlGpuInstanceGetProfileInfoV (*C++ function*), 356
nvmlGpuInstanceGetRemainingCapacity (*C++ function*), 356
nvmlGpuInstanceGetVgpuHeterogeneityMode (*C++ function*), 356
nvmlGpuInstanceGetVgpuSchedulerCapabilities (*C++ function*), 356
nvmlGpuInstanceGetVgpuSchedulerLog (*C++ function*), 356
nvmlGpuInstanceGetVgpuSchedulerState (*C++ function*), 356
nvmlGpuInstanceGetVgpuTypeCreatablePlacements (*C++ function*), 356
nvmlGpuInstanceGetVgpuTypeSupportedPlacements (*C++ function*), 356
nvmlGpuInstanceSetVgpuHeterogeneityMode (*C++ function*), 356
nvmlGpuInstanceSetVgpuSchedulerState (*C++ function*), 356
nvmlGpuOperationMode_t (*C++ type*), 356
nvmlGpuOperationMode_t::NVML_GOM_ALL_ON (*C++ enumerator*), 356
nvmlGpuOperationMode_t::NVML_GOM_COMPUTE (*C++ enumerator*), 356
nvmlGpuOperationMode_t::NVML_GOM_LOW_DP (*C++ enumerator*), 356
nvmlGpuThermalSettings_t (*C++ type*), 356
nvmlGpuThermalSettings_t::count (*C++ member*), 356
nvmlGpuThermalSettings_t::sensor (*C++ member*), 356
nvmlGpuThermalSettings_t::sensor::controller (*C++ member*), 356
nvmlGpuThermalSettings_t::sensor::currentTemp (*C++ member*), 356
nvmlGpuThermalSettings_t::sensor::defaultMaxTemp (*C++ member*), 356
nvmlGpuThermalSettings_t::sensor::defaultMinTemp (*C++ member*), 356
nvmlGpuThermalSettings_t::sensor::target (*C++ member*), 356
nvmlGpuThermalSettings_t::version (*C++ member*), 356
nvmlGridLicensableFeature_t (*C++ type*), 356
nvmlGridLicensableFeature_t::gridLicensableFeatureId (*C++ member*), 356
nvmlGridLicensableFeature_t::featureState (*C++ member*), 356
nvmlGridLicensableFeature_t::licenseExpiry (*C++ member*), 356
nvmlGridLicensableFeature_t::licenseInfo (*C++ member*), 356
nvmlGridLicensableFeature_t::productName (*C++ member*), 356
nvmlGridLicensableFeatures_t (*C++ type*), 356
nvmlGridLicensableFeatures_t::gridLicensableFeature (*C++ member*), 356
nvmlGridLicensableFeatures_t::isGridLicenseSupported (*C++ member*), 356
nvmlGridLicensableFeatures_t::licensableFeaturesCount (*C++ member*), 356
nvmlGridLicensableFeatures_t::version (*C++ member*), 356
nvmlGridLicenseExpiry (*C++ type*), 356
nvmlGridLicenseExpiry::day (*C++ member*), 356
nvmlGridLicenseExpiry::month (*C++ member*), 356
nvmlGridLicenseExpiry::status (*C++ member*), 356
nvmlGridLicenseExpiry::year (*C++ member*), 356
nvmlGridLicenseFeatureCode_t (*C++ type*), 356
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_GAMING (*C++ enumerator*), 356
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_NVAIE (*C++ enumerator*), 356
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_UNKNOWN (*C++ enumerator*), 356
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_VAPP (*C++ enumerator*), 356
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_VCOMPUTE (*C++ enumerator*), 356
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_VDESKTOP (*C++ enumerator*), 356
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_VGPU (*C++ enumerator*), 356
nvmlGridLicenseFeatures_t (*C++ type*), 356
nvmlGridLicenseFeatures_t::gridLicenseExpiry (*C++ member*), 356
nvmlGridLicenseFeatures_t::gridLicenseSupported (*C++ member*), 356
nvmlGridLicenseFeatures_t::licensableFeatures (*C++ member*), 356
nvmlGridLicenseFeatures_t::licensableFeaturesCount (*C++ member*), 356
nvmlGridLicenseFeatures_t::version (*C++ member*), 356
nvmlGridProfileLicenseInfo_t (*C++ type*), 356
nvmlGridProfileLicenseInfo_t::licenseExpiry (*C++ member*), 356
nvmlGridProfileLicenseInfo_t::licenseInfo (*C++ member*), 356
nvmlGridProfileLicenseInfo_t::productName (*C++ member*), 356
nvmlGridProfileLicenseInfo_t::version (*C++ member*), 356
nvmlHostVgpuStatus_t (*C++ type*), 356
nvmlHostVgpuStatus_t::NVML_HOST_VGPU_STATUS_COMPATIBLE (*C++ enumerator*), 356
nvmlHostVgpuStatus_t::NVML_HOST_VGPU_STATUS_INCOMPATIBLE (*C++ enumerator*), 356
nvmlHostVgpuStatus_t::NVML_HOST_VGPU_STATUS_NONE (*C++ enumerator*), 356
nvmlHostVgpuStatus_t::NVML_HOST_VGPU_STATUS_UNKNOWN (*C++ enumerator*), 356
nvmlIndexType_t (*C++ type*), 356
nvmlIndexType_t::NVML_GPU_INDEX (*C++ enumerator*), 356
nvmlIndexType_t::NVML_NVLINK_INDEX (*C++ enumerator*), 356
nvmlInit (*C++ function*), 356
nvmlInit_v2 (*C++ function*), 356
nvmlInitWithFlags (*C++ function*), 356
nvmlInitWithFlags::NVML_INIT_FLAG_NO_ATTACH (*C++ enumerator*), 356
nvmlInitWithFlags::NVML_INIT_FLAG_NO_GPUS (*C++ enumerator*), 356
nvmlMarginTemperature (*C++ type*), 356
nvmlMarginTemperature::marginTemperature (*C++ member*), 356
nvmlMarginTemperature::version (*C++ member*), 356
nvmlMemory_t (*C++ type*), 356
nvmlMemory_t::free (*C++ member*), 356
nvmlMemory_t::total (*C++ member*), 356
nvmlMemory_t::used (*C++ member*), 356
nvmlMemory_v2_t (*C++ type*), 356
nvmlMemory_v2_t::free (*C++ member*), 356
nvmlMemory_v2_t::reserved (*C++ member*), 356
nvmlMemory_v2_t::total (*C++ member*), 356
nvmlMemory_v2_t::used (*C++ member*), 356
nvmlMemory_v2_t::version (*C++ member*), 356
nvmlMemoryAffinity (*C++ function*), 356
nvmlMemoryAffinityScope_t (*C++ type*), 356
nvmlMemoryAffinityScope_t::NVML_AFFINITY_SCOPE_NODE (*C++ enumerator*), 356
nvmlMemoryAffinityScope_t::NVML_AFFINITY_SCOPE_SOCKET (*C++ enumerator*), 356
nvmlMemoryErrorType_t (*C++ type*), 356
nvmlMemoryErrorType_t::NVML_MEMORY_ERROR_TYPE_CORRECTED (*C++ enumerator*), 356
nvmlMemoryErrorType_t::NVML_MEMORY_ERROR_TYPE_UNCORRECTED (*C++ enumerator*), 356
nvmlMemoryErrorType_t::NVML_MEMORY_ERROR_TYPE_UNKNOWN (*C++ enumerator*), 356
nvmlMemoryLocation_t (*C++ type*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_CBU (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_COUNT (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_DEVICE (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_L1_CACHE (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_L2_CACHE (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_REGISTER_FILE (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_SHM (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_SRAM (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_TEXTURE_MEMORY (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_UNKNOWN (*C++ enumerator*), 356
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_VCD (*C++ enumerator*), 356
nvmlMemoryLocationCount (*C++ function*), 356
nvmlMemoryLocationName (*C++ function*), 356
nvmlMemoryLocationType_t (*C++ type*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_DEVICE_MEMORY (*C++ enumerator*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_L1_CACHE (*C++ enumerator*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_L2_CACHE (*C++ enumerator*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_RESERVED (*C++ enumerator*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_TEXTURE_MEMORY (*C++ enumerator*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_UNCACHED (*C++ enumerator*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_UNKNOWN (*C++ enumerator*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_VM_GLOBAL_RESERVED (*C++ enumerator*), 356
nvmlMemoryLocationType_t::NVML_MEMORY_LOCATION_TYPE_VM_SMALL_RESERVED (*C++ enumerator*), 356
nvmlMemoryScope_t (*C++ type*), 356
nvmlMemoryScope_t::NVML_MEMORY_SCOPE_L1 (*C++ enumerator*), 356
nvmlMemoryScope_t::NVML_MEMORY_SCOPE_L2 (*C++ enumerator*), 356
nvmlMigAttributes_t (*C++ type*), 356
nvmlMigAttributes_t::gpuInstanceSliceCount (*C++ member*), 356
nvmlMigAttributes_t::memorySizeMB (*C++ member*), 356
nvmlMigAttributes_t::multiprocessorCount (*C++ member*), 356
nvmlMigAttributes_t::sharedCopyEngineCount (*C++ member*), 356
nvmlMigAttributes_t::computeInstanceSliceCount (*C++ member*), 356
nvmlMigAttributes_t::version (*C++ member*), 356
nvmlMigResourceType_t (*C++ type*), 356
nvmlMigResourceType_t::NVML_MIG_RESOURCE_COPY_ENGINE (*C++ enumerator*), 356
nvmlMigResourceType_t::NVML_MIG_RESOURCE_DECODER (*C++ enumerator*), 356
nvmlMigResourceType_t::NVML_MIG_RESOURCE_ENCODER (*C++ enumerator*), 356
nvmlMigResourceType_t::NVML_MIG_RESOURCE_JPEG (*C++ enumerator*), 356
nvmlMigResourceType_t::NVML_MIG_RESOURCE_MEMORY (*C++ enumerator*), 356
nvmlMigResourceType_t::NVML_MIG_RESOURCE_MULTIPROCESSOR (*C++ enumerator*), 356
nvmlMigResourceType_t::NVML_MIG_RESOURCE_OFA (*C++ enumerator*), 356
nvmlNvLinkCapability_t (*C++ type*), 356
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_COUNT (*C++ enumerator*), 356
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_P2P_ATOMICS (*C++ enumerator*), 356
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_P2P_SUPPORTED (*C++ enumerator*), 356
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_SLI_BRIDGE (*C++ enumerator*), 356
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_SLI_SUPPORTED (*C++ enumerator*), 356
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_SYSMEM_ACCESS (*C++ enumerator*), 356
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_VALID (*C++ enumerator*), 356
nvmlNvLinkErrorCounter_t (*C++ type*), 356
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_COUNT (*C++ enumerator*), 356
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_CRC (*C++ enumerator*), 356
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_ECC (*C++ enumerator*), 356
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_RECOVERY (*C++ enumerator*), 356
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_REPLAY (*C++ enumerator*), 356
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_SYMBOL_BER (*C++ enumerator*), 356
nvmlNvLinkGetBwMode (*C++ function*), 356
nvmlNvLinkGetBwMode_t (*C++ type*), 356
nvmlNvLinkGetBwMode_t::getBwMode (*C++ member*), 356
nvmlNvLinkGetBwMode_t::version (*C++ member*), 356
nvmlNvLinkInfo_t (*C++ type*), 356
nvmlNvLinkInfo_t::isIbSupported (*C++ member*), 356
nvmlNvLinkInfo_t::isIbSupported::isIbSupported (*C++ member*), 356
nvmlNvLinkInfo_t::isIbSupported::version (*C++ member*), 356
nvmlNvLinkInfo_t::version (*C++ member*), 356
nvmlNvLinkPowerThres_t (*C++ type*), 356
nvmlNvLinkPowerThres_t::lowPwrThreshold (*C++ member*), 356
nvmlNvLinkPowerThres_t::version (*C++ member*), 356
nvmlNvLinkSetBwMode (*C++ function*), 356
nvmlNvLinkSetBwMode_t (*C++ type*), 356
nvmlNvLinkSetBwMode_t::setBwMode (*C++ member*), 356
nvmlNvLinkSetBwMode_t::version (*C++ member*), 356
nvmlNvLinkUtilizationControl_t (*C++ type*), 356
nvmlNvLinkUtilizationControl_t::pktfilter (*C++ member*), 356
nvmlNvLinkUtilizationControl_t::units (*C++ member*), 356
nvmlNvLinkUtilizationCountPktTypes_t (*C++ type*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_ALL (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_ATOMIC (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_FLUSH (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_NOP (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_RDATA (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_RESPDATA (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_RESPNODATA (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_RESPPURGED (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_WDATA (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_WDATA_COMPRESSED (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_COUNTER_PKTFILTER_WRITE (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountUnits_t (*C++ type*), 356
nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_BYTES (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_COUNT (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_CYCLES (*C++ enumerator*), 356
nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_PACKETS (*C++ enumerator*), 356
nvmlNvLinkVersion_t (*C++ type*), 356
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_1_0 (*C++ enumerator*), 356
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_2_0 (*C++ enumerator*), 356
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_2_2 (*C++ enumerator*), 356
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_3_0 (*C++ enumerator*), 356
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_3_1 (*C++ enumerator*), 356
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_4_0 (*C++ enumerator*), 356
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_5_0 (*C++ enumerator*), 356
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_6_0 (*C++ enumerator*), 356
nvmlOemBrandType_t (*C++ type*), 356
nvmlOemBrandType_t::NVML_OEM_BRAND_COUNT (*C++ enumerator*), 356
nvmlOemBrandType_t::NVML_OEM_BRAND_HP (*C++ enumerator*), 356
nvmlOemBrandType_t::NVML_OEM_BRAND_INTEL (*C++ enumerator*), 356
nvmlOemBrandType_t::NVML_OEM_BRAND_OTHER (*C++ enumerator*), 356
nvmlOemBrandType_t::NVML_OEM_BRAND_PNY (*C++ enumerator*), 356
nvmlOemBrandType_t::NVML_OEM_BRAND_UNKNOWN (*C++ enumerator*), 356
nvmlPciInfo_t (*C++ type*), 356
nvmlPciInfo_t::busId (*C++ member*), 356
nvmlPciInfo_t::busIdLegacy (*C++ member*), 356
nvmlPciInfo_t::domain (*C++ member*), 356
nvmlPciInfo_t::pciDeviceId (*C++ member*), 356
nvmlPciInfo_t::pciSubSystemId (*C++ member*), 356
nvmlPciInfo_t::reserved0 (*C++ member*), 356
nvmlPciInfo_t::reserved1 (*C++ member*), 356
nvmlPciInfo_t::reserved2 (*C++ member*), 356
nvmlPciInfo_t::reserved3 (*C++ member*), 356
nvmlPciInfoExt (*C++ function*), 356
nvmlPciInfoExt_t (*C++ type*), 356
nvmlPciInfoExt_t::busId (*C++ member*), 356
nvmlPciInfoExt_t::domain (*C++ member*), 356
nvmlPciInfoExt_t::pciDeviceId (*C++ member*), 356
nvmlPciInfoExt_t::pciSubSystemId (*C++ member*), 356
nvmlPciInfoExt_t::version (*C++ member*), 356
nvmlPciInfoExtVersion (*C++ enumerator*), 356
nvmlPciInfoVersion (*C++ enumerator*), 356
nvmlPdi_t (*C++ type*), 356
nvmlPdi_t::mode (*C++ member*), 356
nvmlPdi_t::supported (*C++ member*), 356
nvmlPdi_t::version (*C++ member*), 356
nvmlPdiGet (*C++ function*), 356
nvmlPdiMode_t (*C++ type*), 356
nvmlPdiMode_t::NVML_PDI_MODE_10BIT_TAG_COMPLETER (*C++ enumerator*), 356
nvmlPdiMode_t::NVML_PDI_MODE_10BIT_TAG_REQUESTER (*C++ enumerator*), 356
nvmlPdiMode_t::NVML_PDI_MODE_64BIT (*C++ enumerator*), 356
nvmlPdiMode_t::NVML_PDI_MODE_NO_PDI (*C++ enumerator*), 356
nvmlPdiSupport_t (*C++ type*), 356
nvmlPdiSupport_t::NVML_PDI_SUPPORTED (*C++ enumerator*), 356
nvmlPdiSupport_t::NVML_PDI_UNSUPPORTED (*C++ enumerator*), 356
nvmlPdiSupport_t::NVML_PDI_UNSUPPORTED_MIG_MODE (*C++ enumerator*), 356
nvmlPdiSupport_t::NVML_PDI_UNSUPPORTED_SKU (*C++ enum

(*C++ enumerator*), 365
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L42_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L42_TX_PER_SEC (*C++ enumerator*), 365
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L43_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L43_TX_PER_SEC (*C++ enumerator*), 365
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L44_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L44_TX_PER_SEC (*C++ enumerator*), 365
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L45_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L45_TX_PER_SEC (*C++ enumerator*), 365
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L46_RX_PER_SEC (*C++ enumerator*), 360
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L46_TX_PER_SEC (*C++ enumerator*), 365
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L47_RX_PER_SEC (*C++ enumerator*), 361
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L47_TX_PER_SEC (*C++ enumerator*), 365
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L48_RX_PER_SEC (*C++ enumerator*), 361
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L48_TX_PER_SEC (*C++ enumerator*), 366
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L49_RX_PER_SEC (*C++ enumerator*), 361
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L49_TX_PER_SEC (*C++ enumerator*), 366
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L50_RX_PER_SEC (*C++ enumerator*), 361
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L50_TX_PER_SEC (*C++ enumerator*), 366
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L51_RX_PER_SEC (*C++ enumerator*), 361
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L51_TX_PER_SEC (*C++ enumerator*), 366
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L52_RX_PER_SEC (*C++ enumerator*), 361
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L52_TX_PER_SEC (*C++ enumerator*), 366
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L53_RX_PER_SEC (*C++ enumerator*), 361
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L53_TX_PER_SEC (*C++ enumerator*), 366
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L54_RX_PER_SEC (*C++ enumerator*), 362
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L54_TX_PER_SEC (*C++ enumerator*), 366
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L55_RX_PER_SEC (*C++ enumerator*), 362
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L55_TX_PER_SEC (*C++ enumerator*), 366
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L56_RX_PER_SEC

(*C++ enumerator*), 367
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L56_RX
(*C++ enumerator*), 362
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L56_TX
(*C++ enumerator*), 367
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L57_RX
(*C++ enumerator*), 362
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L57_TX
(*C++ enumerator*), 367
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L58_RX
(*C++ enumerator*), 362
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L58_TX
(*C++ enumerator*), 367
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L59_RX
(*C++ enumerator*), 362
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L59_TX
(*C++ enumerator*), 367
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L60_RX
(*C++ enumerator*), 362
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L60_TX
(*C++ enumerator*), 367
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L61_RX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L61_TX
(*C++ enumerator*), 367
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L62_RX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L62_TX
(*C++ enumerator*), 367
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L63_RX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L63_RX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L63_TX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L63_TX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L64_RX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L64_RX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L64_TX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L64_TX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L65_RX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L65_RX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L65_TX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L65_TX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L66_RX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L66_RX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L66_TX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L66_TX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L67_RX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L67_RX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L67_TX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L67_TX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L68_RX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L68_RX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L68_TX
(*C++ enumerator*), 363
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L68_TX_PERCENT
(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L69_RX
(*C++ enumerator*), 364
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L69_RX_PE

(*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L70_TX_PER_SEC (*C++ enumerator*), 364
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L71_RX_PER_SEC (*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_L71_TX_PER_SEC (*C++ enumerator*), 364
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_TOTAL_RX_PER_SEC (*C++ enumerator*), 368
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVLINK_TOTAL_TX_PER_SEC (*C++ enumerator*), 364
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVOFA0_UTIL (*C++ enumerator*), 369
nvmlGpmMetricId_t::NVML_GPM_METRIC_NVOFA1_UTIL (*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_PCIE_RX_PER_SEC (*C++ enumerator*), 341
nvmlGpmMetricId_t::NVML_GPM_METRIC_PCIE_RX_PER_SEC_BYTES (*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_PCIE_TX_PER_SEC (*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_PCIE_TX_PER_SEC_BYTES (*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_PEERMEM_CACHE_HIT (*C++ enumerator*), 340
nvmlGpmMetricId_t::NVML_GPM_METRIC_PEERMEM_CACHE_MISS (*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_SM_CYCLES_ACTIVE (*C++ enumerator*), 348
nvmlGpmMetricId_t::NVML_GPM_METRIC_SM_CYCLES_ELAPSED (*C++ enumerator*), 354
nvmlGpmMetricId_t::NVML_GPM_METRIC_SM_OCCUPANCY (*C++ enumerator*), 353
nvmlGpmMetricId_t::NVML_GPM_METRIC_SM_UTIL (*C++ enumerator*), 339
nvmlGpmMetricsGet (*C++ function*), 339
nvmlGpmMetricsGet_t (*C++ struct*), 369
nvmlGpmMetricsGet_t::metrics (*C++ member*), 529
nvmlGpmMetricsGet_t::numMetrics (*C++ member*), 530
nvmlGpmMetricsGet_t::sample1 (*C++ member*), 530
nvmlGpmMetricsGet_t::sample2 (*C++ member*), 530
nvmlGpmMetricsGet_t::version (*C++ member*), 530
nvmlGpmMigSampleGet (*C++ function*), 530
nvmlGpmQueryDeviceSupport (*C++ function*), 370
nvmlGpmQueryIfStreamingEnabled (*C++ function*), 370
nvmlGpmSample_t (*C++ type*), 370
nvmlGpmSampleAlloc (*C++ function*), 373
nvmlGpmSampleFree (*C++ function*), 371
nvmlGpmSampleGet (*C++ function*), 371
nvmlGpmSetStreamingEnabled (*C++ function*), 372
nvmlGpmSupport_t (*C++ struct*), 372
nvmlGpmSupport_t::isSupportedDevice (*C++ member*), 530
nvmlGpmSupport_t::version (*C++ member*), 530
nvmlGpuDynamicPstatesInfo_t (*C++ struct*), 530
nvmlGpuDynamicPstatesInfo_t::bIsPresent (*C++ member*), 530
nvmlGpuDynamicPstatesInfo_t::decThreshold (*C++ member*), 531
nvmlGpuDynamicPstatesInfo_t::flags (*C++ member*), 530
nvmlGpuDynamicPstatesInfo_t::incThreshold (*C++ member*), 530
nvmlGpuDynamicPstatesInfo_t::percentage (*C++ member*), 530
nvmlGpuDynamicPstatesInfo_t::utilization (*C++ member*), 531
nvmlGpuFabricClique_v1_t (*C++ struct*), 531
nvmlGpuFabricClique_v1_t::id (*C++ member*), 531
nvmlGpuFabricClique_v1_t::type (*C++ member*), 531
nvmlGpuFabricInfo_t (*C++ struct*), 531
nvmlGpuFabricInfo_t::cliqueId (*C++ member*), 531
nvmlGpuFabricInfo_t::clusterUuid (*C++ member*), 531
nvmlGpuFabricInfo_t::state (*C++ member*), 531
nvmlGpuFabricInfo_t::status (*C++ member*), 531
nvmlGpuFabricInfo_v2 (*C macro*), 268
nvmlGpuFabricInfo_v2_t (*C++ struct*), 532
nvmlGpuFabricInfo_v2_t::cliqueId (*C++*

member), 532
nvmlGpuFabricInfo_v2_t::clusterUuid (*C++ member*), 532
nvmlGpuFabricInfo_v2_t::healthMask (*C++ member*), 532
nvmlGpuFabricInfo_v2_t::state (*C++ member*), 532
nvmlGpuFabricInfo_v2_t::status (*C++ member*), 532
nvmlGpuFabricInfo_v2_t::version (*C++ member*), 532
nvmlGpuFabricInfo_v3 (*C macro*), 268
nvmlGpuFabricInfo_v3_t (*C++ struct*), 532
nvmlGpuFabricInfo_v3_t::cliqueId (*C++ member*), 533
nvmlGpuFabricInfo_v3_t::clusterUuid (*C++ member*), 533
nvmlGpuFabricInfo_v3_t::healthMask (*C++ member*), 533
nvmlGpuFabricInfo_v3_t::healthSummary (*C++ member*), 533
nvmlGpuFabricInfo_v3_t::state (*C++ member*), 533
nvmlGpuFabricInfo_v3_t::status (*C++ member*), 533
nvmlGpuFabricInfo_v3_t::version (*C++ member*), 532
nvmlGpuFabricInfo_v4_t (*C++ struct*), 533
nvmlGpuFabricInfo_v4_t::cliques (*C++ member*), 533
nvmlGpuFabricInfo_v4_t::clusterUuid (*C++ member*), 533
nvmlGpuFabricInfo_v4_t::healthMask (*C++ member*), 533
nvmlGpuFabricInfo_v4_t::healthSummary (*C++ member*), 534
nvmlGpuFabricInfo_v4_t::numCliques (*C++ member*), 533
nvmlGpuFabricInfo_v4_t::state (*C++ member*), 533
nvmlGpuFabricInfo_v4_t::status (*C++ member*), 533
nvmlGpuFabricInfoV_t (*C++ type*), 269
nvmlGpuFabricState_t (*C++ type*), 269
nvmlGpuInstance_t (*C++ type*), 233
nvmlGpuInstanceCreateComputeInstance (*C++ function*), 332
nvmlGpuInstanceCreateComputeInstanceWithPlacement (*C++ function*), 333
nvmlGpuInstanceDestroy (*C++ function*), 334
nvmlGpuInstanceGetActiveVgpus (*C++ function*), 457
nvmlGpuInstanceGetComputeInstanceById (*C++ function*), 334
nvmlGpuInstanceGetComputeInstancePossiblePlacements (*C++ function*), 335
nvmlGpuInstanceGetComputeInstanceProfileInfo (*C++ function*), 335
nvmlGpuInstanceGetComputeInstanceProfileInfoV (*C++ function*), 336
nvmlGpuInstanceGetComputeInstanceRemainingCapacity (*C++ function*), 337
nvmlGpuInstanceGetComputeInstances (*C++ function*), 337
nvmlGpuInstanceGetCreatableVgpus (*C++ function*), 458
nvmlGpuInstanceGetInfo (*C++ function*), 338
nvmlGpuInstanceGetVgpuHeterogeneousMode (*C++ function*), 459
nvmlGpuInstanceGetVgpuSchedulerLog (*C++ function*), 459
nvmlGpuInstanceGetVgpuSchedulerLog_v2 (*C++ function*), 460
nvmlGpuInstanceGetVgpuSchedulerState (*C++ function*), 461
nvmlGpuInstanceGetVgpuSchedulerState_v2 (*C++ function*), 461
nvmlGpuInstanceGetVgpuTypeCreatablePlacements (*C++ function*), 462
nvmlGpuInstanceInfo_t (*C++ struct*), 534
nvmlGpuInstanceInfo_t::device (*C++ member*), 534
nvmlGpuInstanceInfo_t::id (*C++ member*), 534
nvmlGpuInstanceInfo_t::placement (*C++ member*), 534
nvmlGpuInstanceInfo_t::profileId (*C++ member*), 534
nvmlGpuInstancePlacement_t (*C++ struct*), 534
nvmlGpuInstancePlacement_t::size (*C++ member*), 534
nvmlGpuInstancePlacement_t::start (*C++ member*), 534
nvmlGpuInstanceProfileInfo_t (*C++ struct*), 534
nvmlGpuInstanceProfileInfo_t::copyEngineCount (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::decoderCount (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::encoderCount (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::id (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::instanceCount (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::isP2pSupported (*C++ member*), 535

nvmlGpuInstanceProfileInfo_t::jpegCount (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::memorySizeMB (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::multiprocessorCount (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::ofaCount (*C++ member*), 535
nvmlGpuInstanceProfileInfo_t::sliceCount (*C++ member*), 535
nvmlGpuInstanceProfileInfo_v2 (*C macro*), 322
nvmlGpuInstanceProfileInfo_v2_t (*C++ struct*), 535
nvmlGpuInstanceProfileInfo_v2_t::copyEngineCount (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::decoderCount (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::encoderCount (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::id (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::instanceCount (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::isP2pSupported (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::jpegCount (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::memorySizeMB (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::multiprocessorCount (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::name (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::ofaCount (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::sliceCount (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v2_t::version (*C++ member*), 536
nvmlGpuInstanceProfileInfo_v3 (*C macro*), 322
nvmlGpuInstanceProfileInfo_v3_t (*C++ struct*), 536
nvmlGpuInstanceProfileInfo_v3_t::capabilities (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::copyEngineCount (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::decoderCount (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::encoderCount (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::id (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::instanceCount (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::jpegCount (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::memorySizeMB (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::multiprocessorCount (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::name (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::ofaCount (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::sliceCount (*C++ member*), 537
nvmlGpuInstanceProfileInfo_v3_t::version (*C++ member*), 537
nvmlGpuInstanceSetVgpuHeterogeneousMode (*C++ function*), 462
nvmlGpuInstanceSetVgpuSchedulerState (*C++ function*), 463
nvmlGpuInstanceSetVgpuSchedulerState_v2 (*C++ function*), 464
nvmlGpuOperationalEventConfig_v1_t (*C++ struct*), 537
nvmlGpuOperationalEventConfig_v1_t::minLogLevel (*C++ member*), 538
nvmlGpuOperationalEventConfig_v1_t::minSeverity (*C++ member*), 538
nvmlGpuOperationalEventConfig_v1_t::uuid (*C++ member*), 538
nvmlGpuOperationalEventContextType_t (*C++ enum*), 246
nvmlGpuOperationalEventContextType_t::NVML_GPU_OPER (*C++ enumerator*), 246
nvmlGpuOperationalEventContextType_t::NVML_GPU_OPER (*C++ enumerator*), 246
nvmlGpuOperationalEventLogLevel_t (*C++ enum*), 246
nvmlGpuOperationalEventLogLevel_t::NVML_GPU_OPERATI (*C++ enumerator*), 246
nvmlGpuOperationalEventLogLevel_t::NVML_GPU_OPERATI (*C++ enumerator*), 246
nvmlGpuOperationalEventLogLevel_t::NVML_GPU_OPERATI (*C++ enumerator*), 246
nvmlGpuOperationalEventLogLevel_t::NVML_GPU_OPERATI (*C++ enumerator*), 246
nvmlGpuOperationalEventLogLevel_t::NVML_GPU_OPERATI (*C++ enumerator*), 246
nvmlGpuOperationMode_t (*C++ enum*), 87
nvmlGpuOperationMode_t::NVML_GOM_ALL_ON (*C++ enumerator*), 87
nvmlGpuOperationMode_t::NVML_GOM_COMPUTE

(*C++ enumerator*), 87
nvmlGpuOperationMode_t::NVML_GOM_LOW_DP (*C++ enumerator*), 87
nvmlGpuP2PCapsIndex_t (*C++ enum*), 225
nvmlGpuP2PCapsIndex_t::NVML_P2P_CAPS_INDEX_ATOMICS (*C++ enumerator*), 225
nvmlGpuP2PCapsIndex_t::NVML_P2P_CAPS_INDEX_NVLINK (*C++ enumerator*), 225
nvmlGpuP2PCapsIndex_t::NVML_P2P_CAPS_INDEX_PCI (*C++ enumerator*), 225
nvmlGpuP2PCapsIndex_t::NVML_P2P_CAPS_INDEX_PROP (*C++ enumerator*), 225
nvmlGpuP2PCapsIndex_t::NVML_P2P_CAPS_INDEX_READ (*C++ enumerator*), 225
nvmlGpuP2PCapsIndex_t::NVML_P2P_CAPS_INDEX_UNKNOWN (*C++ enumerator*), 225
nvmlGpuP2PCapsIndex_t::NVML_P2P_CAPS_INDEX_WRITE (*C++ enumerator*), 225
nvmlGpuP2PStatus_t (*C++ enum*), 225
nvmlGpuP2PStatus_t::NVML_P2P_STATUS_CHIPSET_NOT_SUPPORTED (*C++ enumerator*), 225
nvmlGpuP2PStatus_t::NVML_P2P_STATUS_CHIPSET_NOT_SUPPORTED (*C++ enumerator*), 225
nvmlGpuP2PStatus_t::NVML_P2P_STATUS_DISABLED_BY_REGKEY (*C++ enumerator*), 226
nvmlGpuP2PStatus_t::NVML_P2P_STATUS_GPU_NOT_SUPPORTED (*C++ enumerator*), 226
nvmlGpuP2PStatus_t::NVML_P2P_STATUS_IOH_TOPOLOGY_NOT_SUPPORTED (*C++ enumerator*), 226
nvmlGpuP2PStatus_t::NVML_P2P_STATUS_NOT_SUPPORTED (*C++ enumerator*), 226
nvmlGpuP2PStatus_t::NVML_P2P_STATUS_OK (*C++ enumerator*), 225
nvmlGpuP2PStatus_t::NVML_P2P_STATUS_UNKNOWN (*C++ enumerator*), 226
nvmlGpuThermalSettings_t (*C++ struct*), 538
nvmlGpuThermalSettings_t::controller (*C++ member*), 538
nvmlGpuThermalSettings_t::count (*C++ member*), 538
nvmlGpuThermalSettings_t::currentTemp (*C++ member*), 538
nvmlGpuThermalSettings_t::defaultMaxTemp (*C++ member*), 538
nvmlGpuThermalSettings_t::defaultMinTemp (*C++ member*), 538
nvmlGpuThermalSettings_t::sensor (*C++ member*), 538
nvmlGpuThermalSettings_t::target (*C++ member*), 538
nvmlGpuTopologyLevel_t (*C++ enum*), 226
nvmlGpuTopologyLevel_t::NVML_TOPOLOGY_HOSTBRIDGE (*C++ enumerator*), 226
nvmlGpuTopologyLevel_t::NVML_TOPOLOGY_INTERNAL (*C++ enumerator*), 226
nvmlGpuTopologyLevel_t::NVML_TOPOLOGY_MULTIPLE (*C++ enumerator*), 226
nvmlGpuTopologyLevel_t::NVML_TOPOLOGY_NODE (*C++ enumerator*), 226
nvmlGpuTopologyLevel_t::NVML_TOPOLOGY_SINGLE (*C++ enumerator*), 226
nvmlGpuTopologyLevel_t::NVML_TOPOLOGY_SYSTEM (*C++ enumerator*), 226
nvmlGpuUptime_v1_t (*C++ struct*), 538
nvmlGpuUptime_v1_t::uptime (*C++ member*), 539
nvmlGpuUtilizationDomainId_t (*C++ enum*), 87
nvmlGpuUtilizationDomainId_t::NVML_GPU_UTILIZATION_DOMAIN_DECODER (*C++ enumerator*), 87
nvmlGpuUtilizationDomainId_t::NVML_GPU_UTILIZATION_DOMAIN_ENCODER (*C++ enumerator*), 87
nvmlGpuUtilizationDomainId_t::NVML_GPU_UTILIZATION_DOMAIN_FB (*C++ enumerator*), 87
nvmlGpuUtilizationDomainId_t::NVML_GPU_UTILIZATION_DOMAIN_GR (*C++ enumerator*), 87
nvmlGpuVirtualizationMode_t (*C++ enum*), 439
nvmlGpuVirtualizationMode_t::NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU (*C++ enumerator*), 439
nvmlGpuVirtualizationMode_t::NVML_GPU_VIRTUALIZATION_MODE_HOST_VSGA (*C++ enumerator*), 439
nvmlGpuVirtualizationMode_t::NVML_GPU_VIRTUALIZATION_MODE_NONE (*C++ enumerator*), 439
nvmlGpuVirtualizationMode_t::NVML_GPU_VIRTUALIZATION_MODE_PASSTHROUGH (*C++ enumerator*), 439
nvmlGpuVirtualizationMode_t::NVML_GPU_VIRTUALIZATION_MODE_VIRTUAL_MACHINE (*C++ enumerator*), 439
nvmlGridLicensableFeature_t (*C++ struct*), 539
nvmlGridLicensableFeature_t::featureCode (*C++ member*), 539
nvmlGridLicensableFeature_t::featureEnabled (*C++ member*), 539
nvmlGridLicensableFeature_t::featureState (*C++ member*), 539
nvmlGridLicensableFeature_t::licenseExpiry (*C++ member*), 539
nvmlGridLicensableFeature_t::licenseInfo (*C++ member*), 539
nvmlGridLicensableFeature_t::productName (*C++ member*), 539
nvmlGridLicensableFeatures_t (*C++ struct*), 539
nvmlGridLicensableFeatures_t::gridLicensableFeatures (*C++ member*), 539
nvmlGridLicensableFeatures_t::isGridLicenseSupported (*C++ member*), 539

nvmlGridLicensableFeatures_t::licensableFeatures (*C++ member*), 539
nvmlGridLicenseExpiry_t (*C++ struct*), 540
nvmlGridLicenseExpiry_t::day (*C++ member*), 540
nvmlGridLicenseExpiry_t::hour (*C++ member*), 540
nvmlGridLicenseExpiry_t::min (*C++ member*), 540
nvmlGridLicenseExpiry_t::month (*C++ member*), 540
nvmlGridLicenseExpiry_t::sec (*C++ member*), 540
nvmlGridLicenseExpiry_t::status (*C++ member*), 540
nvmlGridLicenseExpiry_t::year (*C++ member*), 540
nvmlGridLicenseFeatureCode_t (*C++ enum*), 439
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_COMPUTE (*C++ enumerator*), 440
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_GAMING (*C++ enumerator*), 439
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_NVIDIA_RTX (*C++ enumerator*), 439
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_UNKNOWN (*C++ enumerator*), 439
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_VAPPDEV (*C++ enumerator*), 440
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_VGPU (*C++ enumerator*), 439
nvmlGridLicenseFeatureCode_t::NVML_GRID_LICENSE_FEATURE_CODE_VWORKSTATION (*C++ enumerator*), 439
nvmlHostname_v1_t (*C++ struct*), 540
nvmlHostname_v1_t::value (*C++ member*), 540
nvmlHostVgpuMode_t (*C++ enum*), 440
nvmlHostVgpuMode_t::NVML_HOST_VGPU_MODE_NON_SRIOV (*C++ enumerator*), 440
nvmlHostVgpuMode_t::NVML_HOST_VGPU_MODE_SRIOV (*C++ enumerator*), 440
nvmlHwbcEntry_t (*C++ struct*), 540
nvmlHwbcEntry_t::firmwareVersion (*C++ member*), 541
nvmlHwbcEntry_t::hwbcId (*C++ member*), 541
nvmlInforomObject_t (*C++ enum*), 87
nvmlInforomObject_t::NVML_INFOROM_COUNT (*C++ enumerator*), 88
nvmlInforomObject_t::NVML_INFOROM_DEN (*C++ enumerator*), 88
nvmlInforomObject_t::NVML_INFOROM_ECC (*C++ enumerator*), 88
nvmlInforomObject_t::NVML_INFOROM_OEM (*C++ enumerator*), 88
nvmlInforomObject_t::NVML_INFOROM_POWER (*C++ enumerator*), 88
nvmlInit_v2 (*C++ function*), 312
nvmlInitWithFlags (*C++ function*), 312
nvmlIntNvLinkDeviceType_t (*C++ enum*), 226
nvmlIntNvLinkDeviceType_t::NVML_NVLINK_DEVICE_TYPE (*C++ enumerator*), 226
nvmlIntNvLinkDeviceType_t::NVML_NVLINK_DEVICE_TYPE (*C++ enumerator*), 226
nvmlIntNvLinkDeviceType_t::NVML_NVLINK_DEVICE_TYPE (*C++ enumerator*), 226
nvmlIntNvLinkDeviceType_t::NVML_NVLINK_DEVICE_TYPE (*C++ enumerator*), 226
nvmlLedColor_t (*C++ enum*), 426
nvmlLedColor_t::NVML_LED_COLOR_AMBER (*C++ enumerator*), 426
nvmlLedColor_t::NVML_LED_COLOR_GREEN (*C++ enumerator*), 426
nvmlLedState_t (*C++ struct*), 541
nvmlLedState_t::cause (*C++ member*), 541
nvmlLedState_t::color (*C++ member*), 541
nvmlMarginTemperature_t (*C++ type*), 95
nvmlMarginTemperature_v1 (*C macro*), 83
nvmlMarginTemperature_v1_t (*C++ struct*), 541
nvmlMarginTemperature::marginTemperature (*C++ member*), 541
nvmlMarginTemperature_v1_t::version (*C++ member*), 541
nvmlMask255_t (*C++ struct*), 541
nvmlMask255_t::mask (*C++ member*), 541
nvmlMemory_t (*C++ struct*), 542
nvmlMemory_t::free (*C++ member*), 542
nvmlMemory_t::total (*C++ member*), 542
nvmlMemory_t::used (*C++ member*), 542
nvmlMemory_v2 (*C macro*), 223
nvmlMemory_v2_t (*C++ struct*), 542
nvmlMemory_v2_t::free (*C++ member*), 542
nvmlMemory_v2_t::reserved (*C++ member*), 542
nvmlMemory_v2_t::total (*C++ member*), 542
nvmlMemory_v2_t::used (*C++ member*), 542
nvmlMemory_v2_t::version (*C++ member*), 542
nvmlMemoryErrorType_t (*C++ enum*), 88
nvmlMemoryErrorType_t::NVML_MEMORY_ERROR_TYPE_CORRE (*C++ enumerator*), 88
nvmlMemoryErrorType_t::NVML_MEMORY_ERROR_TYPE_COUNT (*C++ enumerator*), 88
nvmlMemoryErrorType_t::NVML_MEMORY_ERROR_TYPE_UNCOR (*C++ enumerator*), 88
nvmlMemoryLocation_t (*C++ enum*), 88
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_CBU (*C++ enumerator*), 89

nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_COUNT (*C++ enumerator*), 89
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_DEVICE_MEMORY (*C++ enumerator*), 89
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_DRAM (*C++ enumerator*), 89
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_L1_CACHE (*C++ enumerator*), 88
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_L2_CACHE (*C++ enumerator*), 88
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_NVLINK_GPU_REGISTER_FILE (*C++ enumerator*), 89
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_SRAM (*C++ enumerator*), 89
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_TEXTURE_MEMORY (*C++ enumerator*), 89
nvmlMemoryLocation_t::NVML_MEMORY_LOCATION_TEXTURE_SHM (*C++ enumerator*), 89
nvmlNvLinkCapability_t (*C++ enum*), 226
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_COUNT (*C++ enumerator*), 227
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_P2P_ATOMICS (*C++ enumerator*), 227
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_P2P_SUPPORTED (*C++ enumerator*), 227
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_SLI_BRIDGE (*C++ enumerator*), 227
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_SYSMEM_ACCESS (*C++ enumerator*), 227
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_SYSMEM_ATOMICS (*C++ enumerator*), 227
nvmlNvLinkCapability_t::NVML_NVLINK_CAP_VALID (*C++ enumerator*), 227
nvmlNvLinkErrorCounter_t (*C++ enum*), 227
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_COUNT (*C++ enumerator*), 227
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_CRC_DATA (*C++ enumerator*), 227
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_CRC_FLIT (*C++ enumerator*), 227
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_ECC_DATA (*C++ enumerator*), 227
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_RECOVERY (*C++ enumerator*), 227
nvmlNvLinkErrorCounter_t::NVML_NVLINK_ERROR_DL_REPLAY (*C++ enumerator*), 227
nvmlNvlinkFirmwareInfo_t (*C++ struct*), 544
nvmlNvlinkFirmwareInfo_t::firmwareVersion (*C++ member*), 544
nvmlNvlinkFirmwareInfo_t::numValidEntries (*C++ member*), 544
nvmlNvlinkFirmwareVersion_t (*C++ struct*), 544
nvmlNvlinkFirmwareVersion_t::major (*C++ member*), 544
nvmlNvlinkFirmwareVersion_t::minor (*C++ member*), 544
nvmlNvlinkFirmwareVersion_t::subMinor (*C++ member*), 544
nvmlNvlinkFirmwareVersion_t::ucodeType (*C++ member*), 544
nvmlNvlinkGetBwMode_t (*C++ type*), 394
nvmlNvlinkGetBwMode_v1 (*C macro*), 379
nvmlNvlinkGetBwMode_v1_t (*C++ struct*), 544
nvmlNvlinkGetBwMode_v1_t::bIsBest (*C++ member*), 545
nvmlNvlinkGetBwMode_v1_t::bwMode (*C++ member*), 545
nvmlNvlinkGetBwMode_v1_t::version (*C++ member*), 545
nvmlNvlinkGetRemapTableInfo_v1_t (*C++ struct*), 545
nvmlNvlinkGetRemapTableInfo_v1_t::alid (*C++ member*), 545
nvmlNvlinkGetRemapTableInfo_v1_t::flaRemapTabAddr (*C++ member*), 545
nvmlNvlinkGetRemapTableInfo_v1_t::gpaRemapTabAddr (*C++ member*), 545
nvmlNvlinkGetRemapTableInfo_v1_t::remapTabSize (*C++ member*), 545
nvmlNvLinkInfo_t (*C++ type*), 394
nvmlNvLinkInfo_v1 (*C macro*), 379
nvmlNvLinkInfo_v1_t (*C++ struct*), 542
nvmlNvLinkInfo_v1_t::isNvleEnabled (*C++ member*), 542
nvmlNvLinkInfo_v1_t::version (*C++ member*), 542
nvmlNvLinkInfo_v2 (*C macro*), 379
nvmlNvLinkInfo_v2_t (*C++ struct*), 543
nvmlNvLinkInfo_v2_t::firmwareInfo (*C++ member*), 543
nvmlNvLinkInfo_v2_t::isNvleEnabled (*C++ member*), 543
nvmlNvLinkInfo_v2_t::version (*C++ member*), 543
nvmlNvLinkInfo_v3_t (*C++ struct*), 543
nvmlNvLinkInfo_v3_t::firmwareInfo (*C++ member*), 543
nvmlNvLinkInfo_v3_t::isNvleEnabled (*C++ member*), 543
nvmlNvLinkInfo_v3_t::isNvleReady (*C++ member*), 543
nvmlNvlinkLockRemapTable_v1_t (*C++ struct*), 545
nvmlNvlinkLockRemapTable_v1_t::bRemapTableLockDisab (*C++ member*), 545
nvmlNvlinkLockRemapTable_v1_t::bRemapTableLocked (*C++ member*), 545

nvmlNvlinkNvlePrivateInfo_v1_t (*C++ struct*), 545
nvmlNvlinkNvlePrivateInfo_v1_t::data (*C++ member*), 546
nvmlNvlinkNvlePrivateInfo_v1_t::size (*C++ member*), 546
nvmlNvLinkPowerThres_t (*C++ struct*), 543
nvmlNvLinkPowerThres_t::lowPwrThreshold (*C++ member*), 543
nvmlNvlinkSetBwMode_t (*C++ type*), 394
nvmlNvlinkSetBwMode_v1 (*C macro*), 379
nvmlNvlinkSetBwMode_v1_t (*C++ struct*), 546
nvmlNvlinkSetBwMode_v1_t::bSetBest (*C++ member*), 546
nvmlNvlinkSetBwMode_v1_t::bwMode (*C++ member*), 546
nvmlNvlinkSetBwMode_v1_t::version (*C++ member*), 546
nvmlNvlinkSetBwModeAsync_v1_t (*C++ struct*), 546
nvmlNvlinkSetBwModeAsync_v1_t::asyncPollingTimeInMs (*C++ member*), 546
nvmlNvlinkSetBwModeAsync_v1_t::bSetBest (*C++ member*), 546
nvmlNvlinkSetBwModeAsync_v1_t::bwMode (*C++ member*), 546
nvmlNvlinkSetNvleReady_v1_t (*C++ struct*), 546
nvmlNvlinkSetNvleReady_v1_t::bNvleReady (*C++ member*), 547
nvmlNvlinkSetupNvleEncryptionKey_v1_t (*C++ struct*), 547
nvmlNvlinkSetupNvleEncryptionKey_v1_t::localNvleKey (*C++ member*), 547
nvmlNvlinkSetupNvleEncryptionKey_v1_t::localNvleKeyId (*C++ member*), 547
nvmlNvlinkSetupNvleEncryptionKey_v1_t::remoteNvleKey (*C++ member*), 547
nvmlNvlinkSetupNvleEncryptionKey_v1_t::remoteNvleKeyId (*C++ member*), 547
nvmlNvlinkSupportedBwModes_t (*C++ type*), 394
nvmlNvlinkSupportedBwModes_v1 (*C macro*), 380
nvmlNvlinkSupportedBwModes_v1_t (*C++ struct*), 547
nvmlNvlinkSupportedBwModes_v1_t::bwModes (*C++ member*), 547
nvmlNvlinkSupportedBwModes_v1_t::totalBwModes (*C++ member*), 547
nvmlNvlinkSupportedBwModes_v1_t::version (*C++ member*), 547
nvmlNvlinkTelemetrySample_v1_t (*C++ struct*), 547
nvmlNvlinkTelemetrySample_v1_t::linkId (*C++ member*), 548
nvmlNvlinkTelemetrySample_v1_t::nvmlReturn (*C++ member*), 548
nvmlNvlinkTelemetrySample_v1_t::sampleCount (*C++ member*), 548
nvmlNvlinkTelemetrySample_v1_t::samples (*C++ member*), 548
nvmlNvlinkTelemetrySample_v1_t::sampleType (*C++ member*), 548
nvmlNvlinkTelemetrySamples_v1_t (*C++ struct*), 548
nvmlNvlinkTelemetrySamples_v1_t::telemetryCount (*C++ member*), 548
nvmlNvlinkTelemetrySamples_v1_t::telemetrySamples (*C++ member*), 548
nvmlNvlinkTelemetrySampleType_t (*C++ enum*), 380
nvmlNvlinkTelemetrySampleType_t::NVML_NVLINK_TELEME (*C++ enumerator*), 380
nvmlNvlinkTelemetrySampleType_t::NVML_NVLINK_TELEME (*C++ enumerator*), 380
nvmlNvlinkTelemetrySampleType_t::NVML_NVLINK_TELEME (*C++ enumerator*), 380
nvmlNvLinkUtilizationControl_t (*C++ struct*), 544
nvmlNvLinkUtilizationControl_t::pktfilter (*C++ member*), 544
nvmlNvLinkUtilizationControl_t::units (*C++ member*), 544
nvmlNvLinkUtilizationCountPktTypes_t (*C++ enum*), 227
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 227
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 227
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountPktTypes_t::NVML_NVLINK_C (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountUnits_t (*C++ enum*), 228

nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_BYTES (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_COUNT (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_CYCLES (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_PACKETS (*C++ enumerator*), 228
nvmlNvLinkUtilizationCountUnits_t::NVML_NVLINK_COUNTER_UNIT_RESERVED (*C++ enumerator*), 228
nvmlNvlinkVersion_t (*C++ enum*), 380
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_1_0 (*C++ enumerator*), 380
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_2_0 (*C++ enumerator*), 380
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_2_2 (*C++ enumerator*), 380
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_3_0 (*C++ enumerator*), 380
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_3_1 (*C++ enumerator*), 380
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_4_0 (*C++ enumerator*), 380
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_5_0 (*C++ enumerator*), 380
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_6_0 (*C++ enumerator*), 381
nvmlNvlinkVersion_t::NVML_NVLINK_VERSION_INVALID (*C++ enumerator*), 380
nvmlObservedMetrics_t (*C++ struct*), 548
nvmlObservedMetrics_t::bValid (*C++ member*), 548
nvmlObservedMetrics_t::coreRail (*C++ member*), 548
nvmlObservedMetrics_t::fbRail (*C++ member*), 549
nvmlObservedMetrics_t::initialDramclkEst (*C++ member*), 548
nvmlObservedMetrics_t::perfMetrics (*C++ member*), 549
nvmlObservedMetrics_t::tgpPwrTuple (*C++ member*), 549
nvmlOperationalEventSeverity_t (*C++ enum*), 246
nvmlOperationalEventSeverity_t::NVML_OPERATIONAL_EVENT_SEVERITY_ALL (*C++ enumerator*), 247
nvmlOperationalEventSeverity_t::NVML_OPERATIONAL_EVENT_SEVERITY_CORRECTED (*C++ enumerator*), 247
nvmlOperationalEventSeverity_t::NVML_OPERATIONAL_EVENT_SEVERITY_FATAL (*C++ enumerator*), 247
nvmlOperationalEventSeverity_t::NVML_OPERATIONAL_EVENT_SEVERITY_INFORMATIONAL (*C++ enumerator*), 247
nvmlOperationalEventSeverity_t::NVML_OPERATIONAL_EVENT_SEVERITY_RECOVERABLE (*C++ enumerator*), 247
nvmlPageRetirementCause_t (*C++ enum*), 89
nvmlPageRetirementCause_t::NVML_PAGE_RETIREMENT_CAUSE_COUNT (*C++ enumerator*), 89
nvmlPageRetirementCause_t::NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR (*C++ enumerator*), 89
nvmlPageRetirementCause_t::NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS (*C++ enumerator*), 89
nvmlPcieLinkState_t (*C++ enum*), 234
nvmlPcieLinkState_t::NVML_PCIE_LINK_KEEP (*C++ enumerator*), 234
nvmlPcieLinkState_t::NVML_PCIE_LINK_SHUT_DOWN (*C++ enumerator*), 234
nvmlPcieUtilCounter_t (*C++ enum*), 228
nvmlPcieUtilCounter_t::NVML_PCIE_UTIL_COUNT (*C++ enumerator*), 228
nvmlPcieUtilCounter_t::NVML_PCIE_UTIL_RX_BYTES (*C++ enumerator*), 228
nvmlPcieUtilCounter_t::NVML_PCIE_UTIL_TX_BYTES (*C++ enumerator*), 228
nvmlPciInfo_t (*C++ struct*), 552
nvmlPciInfo_t::bus (*C++ member*), 552
nvmlPciInfo_t::busId (*C++ member*), 553
nvmlPciInfo_t::busIdLegacy (*C++ member*), 552
nvmlPciInfo_t::device (*C++ member*), 552
nvmlPciInfo_t::domain (*C++ member*), 552
nvmlPciInfo_t::pciDeviceId (*C++ member*), 552
nvmlPciInfo_t::pciSubSystemId (*C++ member*), 552
nvmlPciInfoExt_t (*C++ type*), 233
nvmlPciInfoExt_v1 (*C macro*), 223
nvmlPciInfoExt_v1_t (*C++ struct*), 551
nvmlPciInfoExt_v1_t::baseClass (*C++ member*), 552
nvmlPciInfoExt_v1_t::bus (*C++ member*), 552
nvmlPciInfoExt_v1_t::busId (*C++ member*), 552
nvmlPciInfoExt_v1_t::device (*C++ member*), 552
nvmlPciInfoExt_v1_t::domain (*C++ member*), 551
nvmlPciInfoExt_v1_t::pciDeviceId (*C++ member*), 552
nvmlPciInfoExt_v1_t::pciSubSystemId (*C++ member*), 552
nvmlPciInfoExt_v1_t::subClass (*C++ member*), 552
nvmlPciInfoExt_v1_t::version (*C++ member*), 551
nvmlPdi_t (*C++ type*), 233
nvmlPdi_v1 (*C macro*), 223
nvmlPdi_v1_t (*C++ struct*), 553

nvmlPdi_v1_t::value (*C++ member*), 553
nvmlPdi_v1_t::version (*C++ member*), 553
nvmlPerfMetricControllerSample_t (*C++ struct*), 553
nvmlPerfMetricControllerSample_t::controllerType (*C++ member*), 553
nvmlPerfMetricControllerSample_t::data (*C++ member*), 553
nvmlPerfMetricControllerSample_t::dlppc2x (*C++ member*), 553
nvmlPerfMetricControllerSample_t::pfpp1x (*C++ member*), 553
nvmlPerfMetricsDlppc2xSample_t (*C++ struct*), 553
nvmlPerfMetricsDlppc2xSample_t::observedMetrics (*C++ member*), 554
nvmlPerfMetricsPfpp1xSample_t (*C++ struct*), 554
nvmlPerfMetricsPfpp1xSample_t::estimatedMetrics (*C++ member*), 554
nvmlPerfMetricsSample_t (*C++ struct*), 554
nvmlPerfMetricsSample_t::controllerData (*C++ member*), 554
nvmlPerfMetricsSample_t::numControllerData (*C++ member*), 554
nvmlPerfMetricsSamples_v1_t (*C++ struct*), 554
nvmlPerfMetricsSamples_v1_t::numSamples (*C++ member*), 554
nvmlPerfMetricsSamples_v1_t::samples (*C++ member*), 554
nvmlPerfPolicyType_t (*C++ enum*), 228
nvmlPerfPolicyType_t::NVML_PERF_POLICY_BOARD_LIMIT (*C++ enumerator*), 229
nvmlPerfPolicyType_t::NVML_PERF_POLICY_COUNT (*C++ enumerator*), 229
nvmlPerfPolicyType_t::NVML_PERF_POLICY_LOW_UTILIZATION (*C++ enumerator*), 229
nvmlPerfPolicyType_t::NVML_PERF_POLICY_POWER (*C++ enumerator*), 229
nvmlPerfPolicyType_t::NVML_PERF_POLICY_RELIABILITY (*C++ enumerator*), 229
nvmlPerfPolicyType_t::NVML_PERF_POLICY_SYNC_BOOST (*C++ enumerator*), 229
nvmlPerfPolicyType_t::NVML_PERF_POLICY_THERMAL (*C++ enumerator*), 229
nvmlPerfPolicyType_t::NVML_PERF_POLICY_TOTAL_APP_CLOCKS (*C++ enumerator*), 229
nvmlPerfPolicyType_t::NVML_PERF_POLICY_TOTAL_BASE_CLOCKS (*C++ enumerator*), 229
nvmlPlatformInfo_t (*C++ type*), 95
nvmlPlatformInfo_v1 (*C macro*), 83
nvmlPlatformInfo_v1_t (*C++ struct*), 555
nvmlPlatformInfo_v1_t::chassisPhysicalSlotNumber (*C++ member*), 555
nvmlPlatformInfo_v1_t::computeSlotIndex (*C++ member*), 555
nvmlPlatformInfo_v1_t::ibGuid (*C++ member*), 555
nvmlPlatformInfo_v1_t::moduleId (*C++ member*), 555
nvmlPlatformInfo_v1_t::nodeIndex (*C++ member*), 555
nvmlPlatformInfo_v1_t::peerType (*C++ member*), 555
nvmlPlatformInfo_v1_t::rackGuid (*C++ member*), 555
nvmlPlatformInfo_v1_t::version (*C++ member*), 555
nvmlPlatformInfo_v2 (*C macro*), 83
nvmlPlatformInfo_v2_t (*C++ struct*), 555
nvmlPlatformInfo_v2_t::chassisSerialNumber (*C++ member*), 556
nvmlPlatformInfo_v2_t::hostId (*C++ member*), 556
nvmlPlatformInfo_v2_t::ibGuid (*C++ member*), 556
nvmlPlatformInfo_v2_t::moduleId (*C++ member*), 556
nvmlPlatformInfo_v2_t::peerType (*C++ member*), 556
nvmlPlatformInfo_v2_t::slotNumber (*C++ member*), 556
nvmlPlatformInfo_v2_t::trayIndex (*C++ member*), 556
nvmlPlatformInfo_v2_t::version (*C++ member*), 556
nvmlPmgrPwrTuple_t (*C++ struct*), 556
nvmlPmgrPwrTuple_t::pwrmW (*C++ member*), 556
nvmlPowerProfileOperation_t (*C++ enum*), 405
nvmlPowerProfileOperation_t::NVML_POWER_PROFILE_OPE (*C++ enumerator*), 405
nvmlPowerProfileOperation_t::NVML_POWER_PROFILE_OPE (*C++ enumerator*), 405
nvmlPowerProfileOperation_t::NVML_POWER_PROFILE_OPE (*C++ enumerator*), 405
nvmlPowerProfileOperation_t::NVML_POWER_PROFILE_OPE (*C++ enumerator*), 405
nvmlPowerProfileType_t (*C++ enum*), 405
nvmlPowerProfileType_t::NVML_POWER_PROFILE_BALANCED (*C++ enumerator*), 405
nvmlPowerProfileType_t::NVML_POWER_PROFILE_COMPUTE (*C++ enumerator*), 405
nvmlPowerProfileType_t::NVML_POWER_PROFILE_DCPCIE (*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_HIGH_THR

(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_HMMA_DENSE
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_HMMA_SPARSE
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_HPC (*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_INFERENCE
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_LLM_INFERENCE
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_LLM_TRAINING
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_LOW_LATENCY_INFERENCE
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_LOW_PRIORITY_BACKGROUND
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MAX
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MAX_P
(*C++ enumerator*), 405
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MAX_Q
(*C++ enumerator*), 405
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MAX_Q_1
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MAX_Q_2
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MAX_Q_3
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MEDIUM
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MEMORY_BOUND
(*C++ enumerator*), 405
nvmlPowerProfileType_t::NVML_POWER_PROFILE_MIG
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_NETWORK
(*C++ enumerator*), 405
nvmlPowerProfileType_t::NVML_POWER_PROFILE_OPTIMAL
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_PCIE
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_SPMC_BALANCED
(*C++ enumerator*), 406
nvmlPowerProfileType_t::NVML_POWER_PROFILE_TRAINING
(*C++ enumerator*), 406
nvmlPowerScopeType_t (*C++ type*), 95
nvmlPowerSmoothingProfile_t (*C++ type*), 414
nvmlPowerSmoothingProfile_v1 (*C macro*), 412
nvmlPowerSmoothingProfile_v1_t (*C++ struct*), 556
nvmlPowerSmoothingProfile_v1_t::paramId (*C++ member*), 557
nvmlPowerSmoothingProfile_v1_t::profileId (*C++ member*), 557
nvmlPowerSmoothingProfile_v1_t::value (*C++ member*), 557
nvmlPowerSmoothingProfile_v1_t::version (*C++ member*), 557
nvmlPowerSmoothingState_t (*C++ type*), 414
nvmlPowerSmoothingState_v1 (*C macro*), 412
nvmlPowerSmoothingState_v1_t (*C++ struct*), 557
nvmlPowerSmoothingState_v1_t::state (*C++ member*), 557
nvmlPowerSmoothingState_v1_t::version (*C++ member*), 557
nvmlPowerSource_t (*C++ type*), 95
nvmlPowerValue_v2 (*C macro*), 483
nvmlPowerValue_v2_t (*C++ struct*), 557
nvmlPowerValue_v2_t::powerScope (*C++ member*), 557
nvmlPowerValue_v2_t::powerValueMw (*C++ member*), 557
nvmlPowerValue_v2_t::version (*C++ member*), 557
nvmlPRMCounter_v1_t (*C++ struct*), 550
nvmlPRMCounter_v1_t::counterId (*C++ member*), 550
nvmlPRMCounter_v1_t::counterValue (*C++ member*), 550
nvmlPRMCounter_v1_t::inData (*C++ member*), 550
nvmlPRMCounterId_t (*C++ enum*), 400
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_NONE (*C++ enumerator*), 400
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PHYSI (*C++ enumerator*), 400
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PHYSI (*C++ enumerator*), 400
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PLR_R (*C++ enumerator*), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PLR_R (*C++ enumerator*), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PLR_R (*C++ enumerator*), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PLR_S (*C++ enumerator*), 402
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PLR_X (*C++ enumerator*), 402
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PLR_X (*C++ enumerator*), 402
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PLR_X (*C++ enumerator*), 402
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_PORTC (*C++ enumerator*), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOV

(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_LAST_SUCCESSFUL_RECOVERY_STEP_A
(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_LAST_SUCCESSFUL_RECOVERY_TIME
(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TIME_BETWEEN_LAST_TWO_RECOVERIE
(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TIME_IN_LAST_HOST_SERDES_FEQ_RE
(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TIME_SINCE_LAST_RECOVERY
(C++ enumerator), 400
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_HOST_SERDES_FEQ_RECOVERY_
(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_HOST_SERDES_FEQ_SUCCESSFU
(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_SUCCESSFUL_RECOVERY_EVENT
(C++ enumerator), 400
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_SUCCESSFUL_RECOVERY_TIME
(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPCNT_RECOVERY_CTRS_TOTAL_TIME_IN_HOST_SERDES_FEQ_R
(C++ enumerator), 401
nvmlPRMCounterId_t::NVML_PRM_COUNTER_ID_PPRM_OPER_RECOVERY
(C++ enumerator), 402
nvmlPRMCounterInput_v1_t (C++ struct), 549
nvmlPRMCounterInput_v1_t::localPort (C++ member), 549
nvmlPRMCounterList_v1_t (C++ struct), 549
nvmlPRMCounterList_v1_t::counters (C++ member), 549
nvmlPRMCounterList_v1_t::numCounters (C++ member), 549
nvmlPRMCounterValue_v1_t (C++ struct), 549
nvmlPRMCounterValue_v1_t::outputType (C++ member), 549
nvmlPRMCounterValue_v1_t::outputValue (C++ member), 550
nvmlPRMCounterValue_v1_t::status (C++ member), 549
nvmlPRMTLV_v1_t (C++ struct), 550
nvmlPRMTLV_v1_t::[anonymous] (C++ member), 550
nvmlPRMTLV_v1_t::dataSize (C++ member), 550
nvmlPRMTLV_v1_t::inData (C++ member), 550
nvmlPRMTLV_v1_t::outData (C++ member), 550
nvmlPRMTLV_v1_t::status (C++ member), 550
nvmlProcessDetail_v1_t (C++ struct), 558
nvmlProcessDetail_v1_t::computeInstanceId (C++ member), 558
nvmlProcessDetail_v1_t::gpuInstanceId (C++ member), 558
nvmlProcessDetail_v1_t::pid (C++ member), 558
nvmlProcessDetail_v1_t::usedGpuCcProtectedMemory (C++ member), 558
nvmlProcessDetail_v1_t::usedGpuMemory (C++ member), 558
nvmlProcessDetailList_t (C++ type), 233
nvmlProcessDetailList_v1_t (C++ struct), 558
nvmlProcessDetailList_v1_t::mode (C++ member), 558
nvmlProcessDetailList_v1_t::numProcArrayEntries (C++ member), 558
nvmlProcessDetailList_v1_t::procArray (C++ member), 558
nvmlProcessDetailList_v1_t::version (C++ member), 558
nvmlProcessesUtilizationInfo_t (C++ type), 233
nvmlProcessesUtilizationInfo_v1 (C++ struct), 560
nvmlProcessesUtilizationInfo_v1_t (C++ struct), 560
nvmlProcessesUtilizationInfo_v1_t::lastSeenTimeStamp (C++ member), 561
nvmlProcessesUtilizationInfo_v1_t::processSamplesCount (C++ member), 561
nvmlProcessesUtilizationInfo_v1_t::procUtilArray (C++ member), 561
nvmlProcessesUtilizationInfo_v1_t::version (C++ member), 561
nvmlProcessInfo_t (C++ struct), 558
nvmlProcessInfo_t::computeInstanceId (C++ member), 559
nvmlProcessInfo_t::gpuInstanceId (C++ member), 559
nvmlProcessInfo_t::pid (C++ member), 559
nvmlProcessInfo_t::usedGpuMemory (C++ member), 559
nvmlProcessInfo_v1_t (C++ struct), 559
nvmlProcessInfo_v1_t::pid (C++ member), 559
nvmlProcessInfo_v1_t::usedGpuMemory (C++ member), 559
nvmlProcessMode_t (C++ enum), 229
nvmlProcessMode_t::NVML_PROCESS_MODE_ALL (C++ enumerator), 229
nvmlProcessMode_t::NVML_PROCESS_MODE_COMPUTE (C++ enumerator), 229
nvmlProcessMode_t::NVML_PROCESS_MODE_GRAPHICS (C++ enumerator), 229
nvmlProcessMode_t::NVML_PROCESS_MODE_MAX (C++ enumerator), 229
nvmlProcessMode_t::NVML_PROCESS_MODE_MPS (C++ enumerator), 229

nvmlProcessUtilizationInfo_v1_t (*C++ struct*), 559
nvmlProcessUtilizationInfo_v1_t::decUtil (*C++ member*), 560
nvmlProcessUtilizationInfo_v1_t::encUtil (*C++ member*), 560
nvmlProcessUtilizationInfo_v1_t::jpgUtil (*C++ member*), 560
nvmlProcessUtilizationInfo_v1_t::memUtil (*C++ member*), 559
nvmlProcessUtilizationInfo_v1_t::ofaUtil (*C++ member*), 560
nvmlProcessUtilizationInfo_v1_t::pid (*C++ member*), 559
nvmlProcessUtilizationInfo_v1_t::smUtil (*C++ member*), 559
nvmlProcessUtilizationInfo_v1_t::timeStamp (*C++ member*), 559
nvmlProcessUtilizationSample_t (*C++ struct*), 560
nvmlProcessUtilizationSample_t::decUtil (*C++ member*), 560
nvmlProcessUtilizationSample_t::encUtil (*C++ member*), 560
nvmlProcessUtilizationSample_t::memUtil (*C++ member*), 560
nvmlProcessUtilizationSample_t::pid (*C++ member*), 560
nvmlProcessUtilizationSample_t::smUtil (*C++ member*), 560
nvmlProcessUtilizationSample_t::timeStamp (*C++ member*), 560
nvmlPstates_t (*C++ enum*), 89
nvmlPstates_t::NVML_PSTATE_0 (*C++ enumerator*), 89
nvmlPstates_t::NVML_PSTATE_1 (*C++ enumerator*), 89
nvmlPstates_t::NVML_PSTATE_2 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_3 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_4 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_5 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_6 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_7 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_8 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_9 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_10 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_11 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_12 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_13 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_14 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_15 (*C++ enumerator*), 90
nvmlPstates_t::NVML_PSTATE_UNKNOWN (*C++ enumerator*), 90
nvmlPSUInfo_t (*C++ struct*), 551
nvmlPSUInfo_t::current (*C++ member*), 551
nvmlPSUInfo_t::power (*C++ member*), 551
nvmlPSUInfo_t::state (*C++ member*), 551
nvmlPSUInfo_t::voltage (*C++ member*), 551
nvmlPwrModelMetricsDlppm1x_t (*C++ struct*), 561
nvmlPwrModelMetricsDlppm1x_t::bValid (*C++ member*), 562
nvmlPwrModelMetricsDlppm1x_t::coreRail (*C++ member*), 562
nvmlPwrModelMetricsDlppm1x_t::fbRail (*C++ member*), 562
nvmlPwrModelMetricsDlppm1x_t::perfMetrics (*C++ member*), 562
nvmlPwrModelMetricsDlppm1x_t::tgpPwrTuple (*C++ member*), 562
nvmlPwrModelMetricsDlppm1xDramclkEstimates_t (*C++ struct*), 561
nvmlPwrModelMetricsDlppm1xDramclkEstimates_t::estim (*C++ member*), 561
nvmlPwrModelMetricsDlppm1xDramclkEstimates_t::numEs (*C++ member*), 561
nvmlPwrModelMetricsDlppm1xPerf_t (*C++ struct*), 561
nvmlPwrModelMetricsDlppm1xPerf_t::perfms (*C++ member*), 561
nvmlPwrModelMetricsPfpp1x_t (*C++ struct*), 562
nvmlPwrModelMetricsPfpp1x_t::bValid (*C++ member*), 562
nvmlPwrModelMetricsPfpp1x_t::estimatedMetrics (*C++ member*), 562
nvmlPwrModelMetricsPfpp1x_t::fmaxAtVmaxPoint (*C++ member*), 562
nvmlPwrModelMetricsPfpp1x_t::maxPerfPerWattPoint (*C++ member*), 562
nvmlPwrModelMetricsPfpp1x_t::numVfPoints (*C++ member*), 562
nvmlPwrModelMetricsPfpp1x_t::tgpHeadroommW (*C++ member*), 562

nvmlPwrModelMetricsSamplePfpp1x_t (*C++ struct*), 563
nvmlPwrModelMetricsSamplePfpp1x_t::estTgpPwrmW (*C++ member*), 563
nvmlPwrModelMetricsSamplePfpp1x_t::freqkHz (*C++ member*), 563
nvmlPwrModelOperatingPointPfpp1x_t (*C++ struct*), 563
nvmlPwrModelOperatingPointPfpp1x_t::freqkHz (*C++ member*), 563
nvmlPwrModelOperatingPointPfpp1x_t::pwrmW (*C++ member*), 563
nvmlRailMetrics_t (*C++ struct*), 563
nvmlRailMetrics_t::freqkHz (*C++ member*), 563
nvmlRailMetrics_t::utilPct (*C++ member*), 563
nvmlRemappedRowsInfo_v2_t (*C++ struct*), 563
nvmlRemappedRowsInfo_v2_t::bFailureOccurred (*C++ member*), 564
nvmlRemappedRowsInfo_v2_t::bPending (*C++ member*), 564
nvmlRemappedRowsInfo_v2_t::corrActiveRemaps (*C++ member*), 564
nvmlRemappedRowsInfo_v2_t::corrInactiveRemaps (*C++ member*), 564
nvmlRemappedRowsInfo_v2_t::uncActiveRemaps (*C++ member*), 564
nvmlRemappedRowsInfo_v2_t::uncInactiveRemaps (*C++ member*), 564
nvmlRepairStatus_t (*C++ type*), 233
nvmlRepairStatus_v1 (*C macro*), 223
nvmlRepairStatus_v1_t (*C++ struct*), 564
nvmlRepairStatus_v1_t::bChannelRepairPending (*C++ member*), 564
nvmlRepairStatus_v1_t::bTpcRepairPending (*C++ member*), 564
nvmlRepairStatus_v1_t::version (*C++ member*), 564
nvmlRestrictedAPI_t (*C++ enum*), 91
nvmlRestrictedAPI_t::NVML_RESTRICTED_API_COUNT (*C++ enumerator*), 91
nvmlRestrictedAPI_t::NVML_RESTRICTED_API_SET_APPLICATION_CLOCKS (*C++ enumerator*), 91
nvmlRestrictedAPI_t::NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS (*C++ enumerator*), 91
nvmlReturn_t (*C++ enum*), 91
nvmlReturn_t::NVML_ERROR_ALREADY_INITIALIZED (*C++ enumerator*), 91
nvmlReturn_t::NVML_ERROR_ARGUMENT_VERSION_MISMATCH (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_CORRUPTED_INFOROM (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_DEPRECATED (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_DRIVER_NOT_LOADED (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_FREQ_NOT_SUPPORTED (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_FUNCTION_NOT_FOUND (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_GPU_IS_LOST (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_GPU_NOT_FOUND (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_IN_USE (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_INSUFFICIENT_POWER (*C++ enumerator*), 91
nvmlReturn_t::NVML_ERROR_INSUFFICIENT_RESOURCES (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_INSUFFICIENT_SIZE (*C++ enumerator*), 91
nvmlReturn_t::NVML_ERROR_INVALID_ARGUMENT (*C++ enumerator*), 91
nvmlReturn_t::NVML_ERROR_INVALID_STATE (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_IRQ_ISSUE (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_LIB_RM_VERSION_MISMATCH (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_LIBRARY_NOT_FOUND (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_MEMORY (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_NO_DATA (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_NO_PERMISSION (*C++ enumerator*), 91
nvmlReturn_t::NVML_ERROR_NOT_FOUND (*C++ enumerator*), 91
nvmlReturn_t::NVML_ERROR_NOT_READY (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_NOT_SUPPORTED (*C++ enumerator*), 91
nvmlReturn_t::NVML_ERROR_OPERATING_SYSTEM (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_RESET_REQUIRED (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_RESET_TYPE_NOT_SUPPORTED (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_TIMEOUT (*C++ enumerator*), 92
nvmlReturn_t::NVML_ERROR_UNINITIALIZED (*C++ enumerator*), 91
nvmlReturn_t::NVML_ERROR_UNKNOWN (*C++ enumerator*), 93
nvmlReturn_t::NVML_ERROR_VGPU_ECC_NOT_SUPPORTED

(*C++ enumerator*), 92
nvmlReturn_t::NVML_SUCCESS (*C++ enumerator*), 91
nvmlRowRemapperHistogramValues_t (*C++ struct*), 564
nvmlRowRemapperHistogramValues_t::high (*C++ member*), 565
nvmlRowRemapperHistogramValues_t::low (*C++ member*), 565
nvmlRowRemapperHistogramValues_t::max (*C++ member*), 565
nvmlRowRemapperHistogramValues_t::none (*C++ member*), 565
nvmlRowRemapperHistogramValues_t::partial (*C++ member*), 565
nvmlRusdSettings_v1 (*C macro*), 83
nvmlRusdSettings_v1_t (*C++ struct*), 565
nvmlRusdSettings_v1_t::pollMask (*C++ member*), 565
nvmlRusdSettings_v1_t::version (*C++ member*), 565
nvmlSample_t (*C++ struct*), 565
nvmlSample_t::sampleValue (*C++ member*), 565
nvmlSample_t::timeStamp (*C++ member*), 565
nvmlSamplingType_t (*C++ enum*), 230
nvmlSamplingType_t::NVML_DEC_UTILIZATION_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_ENC_UTILIZATION_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_GPU_UTILIZATION_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_JPG_UTILIZATION_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_MEMORY_CLK_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_MEMORY_UTILIZATION_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_MODULE_POWER_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_OFA_UTILIZATION_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_PROCESSOR_CLK_SAMPLES (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_SAMPLINGTYPE_COUNT (*C++ enumerator*), 230
nvmlSamplingType_t::NVML_TOTAL_POWER_SAMPLES (*C++ enumerator*), 230
nvmlSetMemoryLimits_v1_t (*C++ struct*), 565
nvmlSetMemoryLimits_v1_t::hardLimit (*C++ member*), 566
nvmlSetMemoryLimits_v1_t::nameSpace (*C++ member*), 566
nvmlSetMemoryLimits_v1_t::softLimit (*C++ member*), 566
nvmlSetVgpuVersion (*C++ function*), 486
nvmlShutdown (*C++ function*), 313
nvmlSystemConfComputeSettings_t (*C++ type*), 52
nvmlSystemConfComputeSettings_v1 (*C macro*), 52
nvmlSystemConfComputeSettings_v1_t (*C++ struct*), 566
nvmlSystemConfComputeSettings_v1_t::ccFeature (*C++ member*), 566
nvmlSystemConfComputeSettings_v1_t::devToolsMode (*C++ member*), 566
nvmlSystemConfComputeSettings_v1_t::environment (*C++ member*), 566
nvmlSystemConfComputeSettings_v1_t::multiGpuMode (*C++ member*), 566
nvmlSystemConfComputeSettings_v1_t::version (*C++ member*), 566
nvmlSystemConfComputeSettings_v2 (*C macro*), 52
nvmlSystemConfComputeSettings_v2_t (*C++ struct*), 566
nvmlSystemConfComputeSettings_v2_t::ccFeature (*C++ member*), 567
nvmlSystemConfComputeSettings_v2_t::devToolsMode (*C++ member*), 567
nvmlSystemConfComputeSettings_v2_t::environment (*C++ member*), 567
nvmlSystemConfComputeSettings_v2_t::multiGpuMode (*C++ member*), 567
nvmlSystemConfComputeSettings_v2_t::tdispState (*C++ member*), 567
nvmlSystemConfComputeSettings_v2_t::version (*C++ member*), 567
nvmlSystemDriverBranchInfo_t (*C++ type*), 420
nvmlSystemDriverBranchInfo_v1 (*C macro*), 415
nvmlSystemDriverBranchInfo_v1_t (*C++ struct*), 567
nvmlSystemDriverBranchInfo_v1_t::branch (*C++ member*), 567
nvmlSystemDriverBranchInfo_v1_t::version (*C++ member*), 567
nvmlSystemEventData_v1_t (*C++ struct*), 567
nvmlSystemEventData_v1_t::eventType (*C++ member*), 567
nvmlSystemEventData_v1_t::gpuId (*C++ member*), 567
nvmlSystemEventSet_t (*C++ type*), 260
nvmlSystemEventSetCreate (*C++ function*), 257
nvmlSystemEventSetCreateRequest_t (*C++*

type), 260
nvmlSystemEventSetCreateRequest_v1 (*C macro*), 245
nvmlSystemEventSetCreateRequest_v1_t (*C++ struct*), 567
nvmlSystemEventSetCreateRequest_v1_t::set (*C++ member*), 568
nvmlSystemEventSetCreateRequest_v1_t::version (*C++ member*), 568
nvmlSystemEventSetFree (*C++ function*), 258
nvmlSystemEventSetFreeRequest_t (*C++ type*), 260
nvmlSystemEventSetFreeRequest_v1 (*C macro*), 245
nvmlSystemEventSetFreeRequest_v1_t (*C++ struct*), 568
nvmlSystemEventSetFreeRequest_v1_t::set (*C++ member*), 568
nvmlSystemEventSetFreeRequest_v1_t::version (*C++ member*), 568
nvmlSystemEventSetWait (*C++ function*), 258
nvmlSystemEventSetWaitRequest_t (*C++ type*), 260
nvmlSystemEventSetWaitRequest_v1 (*C macro*), 245
nvmlSystemEventSetWaitRequest_v1_t (*C++ struct*), 568
nvmlSystemEventSetWaitRequest_v1_t::data (*C++ member*), 568
nvmlSystemEventSetWaitRequest_v1_t::dataSize (*C++ member*), 568
nvmlSystemEventSetWaitRequest_v1_t::numEvent (*C++ member*), 568
nvmlSystemEventSetWaitRequest_v1_t::set (*C++ member*), 568
nvmlSystemEventSetWaitRequest_v1_t::timeoutms (*C++ member*), 568
nvmlSystemEventSetWaitRequest_v1_t::version (*C++ member*), 568
nvmlSystemEventTypeCount (*C macro*), 245
nvmlSystemEventTypeGpuDriverBind (*C macro*), 245
nvmlSystemEventTypeGpuDriverUnbind (*C macro*), 245
nvmlSystemGetConfComputeCapabilities (*C++ function*), 212
nvmlSystemGetConfComputeGpusReadyState (*C++ function*), 212
nvmlSystemGetConfComputeKeyRotationThresholdInfo (*C++ function*), 213
nvmlSystemGetConfComputeSettings (*C++ function*), 213
nvmlSystemGetConfComputeSettings_v2 (*C++ function*), 213
nvmlSystemGetConfComputeState (*C++ function*), 214
nvmlSystemGetCPER_v1 (*C++ function*), 415
nvmlSystemGetCudaDriverVersion (*C++ function*), 416
nvmlSystemGetCudaDriverVersion_v2 (*C++ function*), 416
nvmlSystemGetDriverBranch (*C++ function*), 417
nvmlSystemGetDriverVersion (*C++ function*), 417
nvmlSystemGetHicVersion (*C++ function*), 418
nvmlSystemGetNvlinkBwMode (*C++ function*), 393
nvmlSystemGetNVMLVersion (*C++ function*), 418
nvmlSystemGetProcessName (*C++ function*), 418
nvmlSystemGetTopologyGpuSet (*C++ function*), 419
nvmlSystemRegisterEventRequest_t (*C++ type*), 260
nvmlSystemRegisterEventRequest_v1 (*C macro*), 245
nvmlSystemRegisterEventRequest_v1_t (*C++ struct*), 569
nvmlSystemRegisterEventRequest_v1_t::eventTypes (*C++ member*), 569
nvmlSystemRegisterEventRequest_v1_t::set (*C++ member*), 569
nvmlSystemRegisterEventRequest_v1_t::version (*C++ member*), 569
nvmlSystemRegisterEvents (*C++ function*), 259
nvmlSystemSetConfComputeGpusReadyState (*C++ function*), 214
nvmlSystemSetConfComputeKeyRotationThresholdInfo (*C++ function*), 215
nvmlSystemSetNvlinkBwMode (*C++ function*), 393
nvmlTemperature_t (*C++ type*), 215
nvmlTemperature_v1 (*C macro*), 112
nvmlTemperature_v1_t (*C++ struct*), 569
nvmlTemperature_v1_t::sensorType (*C++ member*), 569
nvmlTemperature_v1_t::temperature (*C++ member*), 569
nvmlTemperature_v1_t::version (*C++ member*), 569
nvmlTemperatureSensors_t (*C++ enum*), 93
nvmlTemperatureSensors_t::NVML_TEMPERATURE_COUNT (*C++ enumerator*), 93
nvmlTemperatureSensors_t::NVML_TEMPERATURE_GPU (*C++ enumerator*), 93

nvmlTemperatureSensors_t::NVML_TEMPERATURE_GPU_MAX (*C++ enumerator*), 93
nvmlTemperatureThresholds_t (*C++ enum*), 93
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_CURR (*C++ enumerator*), 94
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_MAX (*C++ enumerator*), 94
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_ACOUSTIC_MIN (*C++ enumerator*), 94
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_COUNT (*C++ enumerator*), 94
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_GPS_CURR (*C++ enumerator*), 94
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_GPU_MAX (*C++ enumerator*), 94
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_MEM_MAX (*C++ enumerator*), 94
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_SHUTDOWN (*C++ enumerator*), 93
nvmlTemperatureThresholds_t::NVML_TEMPERATURE_THRESHOLD_SLOWDOWN (*C++ enumerator*), 93
nvmlThermalController_t (*C++ enum*), 230
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_ADM1032 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_ADT7461 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_ADT7473 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_EMC2103 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_G781 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_GPU_INTERNAL (*C++ enumerator*), 230
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_LM63 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_LM89 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_LM90 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_MAX1617 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_MAX6649 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_MAX6657 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_NONE (*C++ enumerator*), 230
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_PCT2075 (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_SBTSI (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_UNKNOWN (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_VEGA_CURR (*C++ enumerator*), 231
nvmlThermalController_t::NVML_THERMAL_CONTROLLER_VEGA_MAX (*C++ enumerator*), 231
nvmlThermalTarget_t (*C++ enum*), 231
nvmlThermalTarget_t::NVML_THERMAL_TARGET_ALL (*C++ enumerator*), 232
nvmlThermalTarget_t::NVML_THERMAL_TARGET_BOARD (*C++ enumerator*), 232
nvmlThermalTarget_t::NVML_THERMAL_TARGET_GPU (*C++ enumerator*), 231
nvmlThermalTarget_t::NVML_THERMAL_TARGET_MEMORY (*C++ enumerator*), 232
nvmlThermalTarget_t::NVML_THERMAL_TARGET_NONE (*C++ enumerator*), 231
nvmlThermalTarget_t::NVML_THERMAL_TARGET_POWER_SUPPLY (*C++ enumerator*), 232
nvmlThermalTarget_t::NVML_THERMAL_TARGET_UNKNOWN (*C++ enumerator*), 232
nvmlThermalTarget_t::NVML_THERMAL_TARGET_VCD_BOARD (*C++ enumerator*), 232
nvmlThermalTarget_t::NVML_THERMAL_TARGET_VCD_INLET (*C++ enumerator*), 232
nvmlThermalTarget_t::NVML_THERMAL_TARGET_VCD_OUTLET (*C++ enumerator*), 232
nvmlUnit_t (*C++ type*), 426
nvmlUnitFanInfo_t (*C++ struct*), 570
nvmlUnitFanInfo_t::speed (*C++ member*), 570
nvmlUnitFanInfo_t::state (*C++ member*), 570
nvmlUnitFanSpeeds_t (*C++ struct*), 570
nvmlUnitFanSpeeds_t::count (*C++ member*), 570
nvmlUnitFanSpeeds_t::fans (*C++ member*), 570
nvmlUnitGetCount (*C++ function*), 421
nvmlUnitGetDevices (*C++ function*), 421
nvmlUnitGetFanSpeedInfo (*C++ function*), 422
nvmlUnitGetHandleByIndex (*C++ function*), 423
nvmlUnitGetLedState (*C++ function*), 423
nvmlUnitGetPsuInfo (*C++ function*), 423
nvmlUnitGetTemperature (*C++ function*), 424
nvmlUnitGetUnitInfo (*C++ function*), 424
nvmlUnitInfo_t (*C++ struct*), 570
nvmlUnitInfo_t::firmwareVersion (*C++ member*), 570
nvmlUnitInfo_t::hwbcCount (*C++ member*), 570
nvmlUnitInfo_t::name (*C++ member*), 570
nvmlUnitInfo_t::serial (*C++ member*), 570

nvmlUnitSetLedState (*C++ function*), 420
nvmlUnrepairableMemoryStatus_v1_t (*C++ struct*), 571
nvmlUnrepairableMemoryStatus_v1_t::bUnrepairableMemory (*C++ member*), 571
nvmlUtilization_t (*C++ struct*), 571
nvmlUtilization_t::gpu (*C++ member*), 571
nvmlUtilization_t::memory (*C++ member*), 571
nvmlUUID_t (*C++ type*), 233
nvmlUUID_v1 (*C macro*), 224
nvmlUUID_v1_t (*C++ struct*), 569
nvmlUUID_v1_t::type (*C++ member*), 569
nvmlUUID_v1_t::value (*C++ member*), 569
nvmlUUID_v1_t::version (*C++ member*), 569
nvmlUUIDType_t (*C++ enum*), 232
nvmlUUIDType_t::NVML_UUID_TYPE_ASCII (*C++ enumerator*), 232
nvmlUUIDType_t::NVML_UUID_TYPE_BINARY (*C++ enumerator*), 232
nvmlUUIDType_t::NVML_UUID_TYPE_NONE (*C++ enumerator*), 232
nvmlUUIDValue_t (*C++ union*), 599
nvmlUUIDValue_t::bytes (*C++ member*), 599
nvmlUUIDValue_t::str (*C++ member*), 599
nvmlValue_t (*C++ union*), 599
nvmlValue_t::dVal (*C++ member*), 599
nvmlValue_t::siVal (*C++ member*), 599
nvmlValue_t::sllVal (*C++ member*), 600
nvmlValue_t::uiVal (*C++ member*), 599
nvmlValue_t::ullVal (*C++ member*), 600
nvmlValue_t::ulVal (*C++ member*), 599
nvmlValue_t::usVal (*C++ member*), 600
nvmlValueType_t (*C++ enum*), 232
nvmlValueType_t::NVML_VALUE_TYPE_COUNT (*C++ enumerator*), 233
nvmlValueType_t::NVML_VALUE_TYPE_DOUBLE (*C++ enumerator*), 232
nvmlValueType_t::NVML_VALUE_TYPE_SIGNED_INT (*C++ enumerator*), 233
nvmlValueType_t::NVML_VALUE_TYPE_SIGNED_LONG_LONG (*C++ enumerator*), 233
nvmlValueType_t::NVML_VALUE_TYPE_UNSIGNED_INT (*C++ enumerator*), 232
nvmlValueType_t::NVML_VALUE_TYPE_UNSIGNED_LONG (*C++ enumerator*), 233
nvmlValueType_t::NVML_VALUE_TYPE_UNSIGNED_LONG_LONG (*C++ enumerator*), 233
nvmlValueType_t::NVML_VALUE_TYPE_UNSIGNED_SHORT (*C++ enumerator*), 233
nvmlVgpuCapability_t (*C++ enum*), 440
nvmlVgpuCapability_t::NVML_VGPU_CAP_COUNT (*C++ enumerator*), 440
nvmlVgpuCapability_t::NVML_VGPU_CAP_EXCLUSIVE_SIZE (*C++ enumerator*), 440
nvmlVgpuCapability_t::NVML_VGPU_CAP_EXCLUSIVE_TYPE (*C++ enumerator*), 440
nvmlVgpuCapability_t::NVML_VGPU_CAP_GPUDIRECT (*C++ enumerator*), 440
nvmlVgpuCapability_t::NVML_VGPU_CAP_MULTI_VGPU_EXCL (*C++ enumerator*), 440
nvmlVgpuCapability_t::NVML_VGPU_CAP_NVLINK_P2P (*C++ enumerator*), 440
nvmlVgpuCreatablePlacementInfo_t (*C++ type*), 449
nvmlVgpuCreatablePlacementInfo_v1 (*C macro*), 447
nvmlVgpuCreatablePlacementInfo_v1_t (*C++ struct*), 571
nvmlVgpuCreatablePlacementInfo_v1_t::count (*C++ member*), 571
nvmlVgpuCreatablePlacementInfo_v1_t::placementIds (*C++ member*), 571
nvmlVgpuCreatablePlacementInfo_v1_t::placementSize (*C++ member*), 571
nvmlVgpuCreatablePlacementInfo_v1_t::version (*C++ member*), 571
nvmlVgpuCreatablePlacementInfo_v1_t::vgpuTypeId (*C++ member*), 571
nvmlVgpuDriverCapability_t (*C++ enum*), 440
nvmlVgpuDriverCapability_t::NVML_VGPU_DRIVER_CAP_CO (*C++ enumerator*), 441
nvmlVgpuDriverCapability_t::NVML_VGPU_DRIVER_CAP_HE (*C++ enumerator*), 440
nvmlVgpuDriverCapability_t::NVML_VGPU_DRIVER_CAP_WA (*C++ enumerator*), 441
nvmlVgpuGuestInfoState_t (*C++ enum*), 441
nvmlVgpuGuestInfoState_t::NVML_VGPU_INSTANCE_GUEST (*C++ enumerator*), 441
nvmlVgpuGuestInfoState_t::NVML_VGPU_INSTANCE_GUEST (*C++ enumerator*), 441
nvmlVgpuHeterogeneousMode_t (*C++ type*), 449
nvmlVgpuHeterogeneousMode_v1 (*C macro*), 447
nvmlVgpuHeterogeneousMode_v1_t (*C++ struct*), 572
nvmlVgpuHeterogeneousMode_v1_t::mode (*C++ member*), 572
nvmlVgpuHeterogeneousMode_v1_t::version (*C++ member*), 572
nvmlVgpuInstance_t (*C++ type*), 449
nvmlVgpuInstanceClearAccountingPids (*C++ function*), 493
nvmlVgpuInstanceGetAccountingMode (*C++ function*), 494
nvmlVgpuInstanceGetAccountingPids (*C++

function), 495
nvmlVgpuInstanceGetAccountingStats (C++ function), 496
nvmlVgpuInstanceGetEccMode (C++ function), 464
nvmlVgpuInstanceGetEncoderCapacity (C++ function), 465
nvmlVgpuInstanceGetEncoderSessions (C++ function), 465
nvmlVgpuInstanceGetEncoderStats (C++ function), 466
nvmlVgpuInstanceGetFBCSessions (C++ function), 466
nvmlVgpuInstanceGetFBCStats (C++ function), 467
nvmlVgpuInstanceGetFbUsage (C++ function), 468
nvmlVgpuInstanceGetFrameRateLimit (C++ function), 468
nvmlVgpuInstanceGetGpuInstanceId (C++ function), 469
nvmlVgpuInstanceGetGpuPciId (C++ function), 469
nvmlVgpuInstanceGetLicenseInfo_v2 (C++ function), 497
nvmlVgpuInstanceGetLicenseStatus (C++ function), 470
nvmlVgpuInstanceGetMdevUUID (C++ function), 470
nvmlVgpuInstanceGetMetadata (C++ function), 487
nvmlVgpuInstanceGetPlacementId (C++ function), 433
nvmlVgpuInstanceGetRuntimeStateSize (C++ function), 433
nvmlVgpuInstanceGetType (C++ function), 471
nvmlVgpuInstanceGetUUID (C++ function), 471
nvmlVgpuInstanceGetVmDriverVersion (C++ function), 472
nvmlVgpuInstanceGetVmID (C++ function), 472
nvmlVgpuInstanceSetEncoderCapacity (C++ function), 473
nvmlVgpuInstancesUtilizationInfo_t (C++ type), 449
nvmlVgpuInstancesUtilizationInfo_v1 (C macro), 447
nvmlVgpuInstancesUtilizationInfo_v1_t (C++ struct), 573
nvmlVgpuInstancesUtilizationInfo_v1_t::lastSeenTimeStamp (C++ member), 573
nvmlVgpuInstancesUtilizationInfo_v1_t::sampleValType (C++ member), 573
nvmlVgpuInstancesUtilizationInfo_v1_t::vgpuCount (C++ member), 573
nvmlVgpuInstancesUtilizationInfo_v1_t::vgpuInstance (C++ member), 573
nvmlVgpuInstancesUtilizationInfo_v1_t::vgpuUtilArray (C++ member), 573
nvmlVgpuInstanceUtilizationInfo_v1_t (C++ struct), 572
nvmlVgpuInstanceUtilizationInfo_v1_t::decUtil (C++ member), 572
nvmlVgpuInstanceUtilizationInfo_v1_t::encUtil (C++ member), 572
nvmlVgpuInstanceUtilizationInfo_v1_t::jpgUtil (C++ member), 572
nvmlVgpuInstanceUtilizationInfo_v1_t::memUtil (C++ member), 572
nvmlVgpuInstanceUtilizationInfo_v1_t::ofaUtil (C++ member), 572
nvmlVgpuInstanceUtilizationInfo_v1_t::smUtil (C++ member), 572
nvmlVgpuInstanceUtilizationInfo_v1_t::timeStamp (C++ member), 572
nvmlVgpuInstanceUtilizationInfo_v1_t::vgpuInstance (C++ member), 572
nvmlVgpuInstanceUtilizationSample_t (C++ struct), 573
nvmlVgpuInstanceUtilizationSample_t::decUtil (C++ member), 573
nvmlVgpuInstanceUtilizationSample_t::encUtil (C++ member), 573
nvmlVgpuInstanceUtilizationSample_t::memUtil (C++ member), 573
nvmlVgpuInstanceUtilizationSample_t::smUtil (C++ member), 573
nvmlVgpuInstanceUtilizationSample_t::timeStamp (C++ member), 573
nvmlVgpuInstanceUtilizationSample_t::vgpuInstance (C++ member), 573
nvmlVgpuLicenseExpiry_t (C++ struct), 574
nvmlVgpuLicenseExpiry_t::day (C++ member), 574
nvmlVgpuLicenseExpiry_t::hour (C++ member), 574
nvmlVgpuLicenseExpiry_t::min (C++ member), 574
nvmlVgpuLicenseExpiry_t::month (C++ member), 574
nvmlVgpuLicenseExpiry_t::sec (C++ member), 574
nvmlVgpuLicenseExpiry_t::status (C++ member), 574
nvmlVgpuLicenseExpiry_t::year (C++ member), 574
nvmlVgpuLicenseInfo_t (C++ struct), 574
nvmlVgpuLicenseInfo_t::currentState (C++ member), 574

nvmlVgpuLicenseInfo_t::isLicensed (*C++ member*), 574
nvmlVgpuLicenseInfo_t::licenseExpiry (*C++ member*), 574
nvmlVgpuMetadata_t (*C++ struct*), 575
nvmlVgpuMetadata_t::guestDriverVersion (*C++ member*), 575
nvmlVgpuMetadata_t::guestInfoState (*C++ member*), 575
nvmlVgpuMetadata_t::guestVgpuVersion (*C++ member*), 575
nvmlVgpuMetadata_t::hostDriverVersion (*C++ member*), 575
nvmlVgpuMetadata_t::opaqueData (*C++ member*), 575
nvmlVgpuMetadata_t::opaqueDataSize (*C++ member*), 575
nvmlVgpuMetadata_t::reserved (*C++ member*), 575
nvmlVgpuMetadata_t::revision (*C++ member*), 575
nvmlVgpuMetadata_t::version (*C++ member*), 575
nvmlVgpuMetadata_t::vgpuVirtualizationCaps (*C++ member*), 575
nvmlVgpuPgpuCompatibility_t (*C++ struct*), 575
nvmlVgpuPgpuCompatibility_t::compatibilityLimitCode (*C++ member*), 576
nvmlVgpuPgpuCompatibility_t::vgpuVmCompatibility (*C++ member*), 575
nvmlVgpuPgpuCompatibilityLimitCode_t (*C++ enum*), 481
nvmlVgpuPgpuCompatibilityLimitCode_t::NVML_VGPU_COMPATIBILITY_LIMIT_GPU (*C++ enumerator*), 481
nvmlVgpuPgpuCompatibilityLimitCode_t::NVML_VGPU_COMPATIBILITY_LIMIT_GUEST_DRIVER (*C++ enumerator*), 481
nvmlVgpuPgpuCompatibilityLimitCode_t::NVML_VGPU_COMPATIBILITY_LIMIT_HOST_DRIVER (*C++ enumerator*), 481
nvmlVgpuPgpuCompatibilityLimitCode_t::NVML_VGPU_COMPATIBILITY_LIMIT_NONE (*C++ enumerator*), 481
nvmlVgpuPgpuCompatibilityLimitCode_t::NVML_VGPU_COMPATIBILITY_LIMIT_OTHER (*C++ enumerator*), 481
nvmlVgpuPgpuMetadata_t (*C++ struct*), 576
nvmlVgpuPgpuMetadata_t::hostDriverVersion (*C++ member*), 576
nvmlVgpuPgpuMetadata_t::hostSupportedVgpuRange (*C++ member*), 576
nvmlVgpuPgpuMetadata_t::opaqueData (*C++ member*), 576
nvmlVgpuPgpuMetadata_t::opaqueDataSize (*C++ member*), 576
nvmlVgpuPgpuMetadata_t::pgpuVirtualizationCaps (*C++ member*), 576
nvmlVgpuPgpuMetadata_t::reserved (*C++ member*), 576
nvmlVgpuPgpuMetadata_t::revision (*C++ member*), 576
nvmlVgpuPgpuMetadata_t::version (*C++ member*), 576
nvmlVgpuPlacementId_t (*C++ type*), 449
nvmlVgpuPlacementId_v1 (*C macro*), 447
nvmlVgpuPlacementId_v1_t (*C++ struct*), 576
nvmlVgpuPlacementId_v1_t::placementId (*C++ member*), 577
nvmlVgpuPlacementId_v1_t::version (*C++ member*), 577
nvmlVgpuPlacementList_t (*C++ type*), 449
nvmlVgpuPlacementList_v1 (*C macro*), 447
nvmlVgpuPlacementList_v1_t (*C++ struct*), 577
nvmlVgpuPlacementList_v1_t::count (*C++ member*), 577
nvmlVgpuPlacementList_v1_t::placementIds (*C++ member*), 577
nvmlVgpuPlacementList_v1_t::placementSize (*C++ member*), 577
nvmlVgpuPlacementList_v1_t::version (*C++ member*), 577
nvmlVgpuPlacementList_v2 (*C macro*), 447
nvmlVgpuPlacementList_v2_t (*C++ struct*), 577
nvmlVgpuPlacementList_v2_t::count (*C++ member*), 577
nvmlVgpuPlacementList_v2_t::mode (*C++ member*), 577
nvmlVgpuPlacementList_v2_t::placementIds (*C++ member*), 577
nvmlVgpuPlacementList_v2_t::placementSize (*C++ member*), 577
nvmlVgpuPlacementList_v2_t::version (*C++ member*), 577
nvmlVgpuProcessesUtilizationInfo_t (*C++ type*), 449
nvmlVgpuProcessesUtilizationInfo_v1 (*C macro*), 448
nvmlVgpuProcessesUtilizationInfo_v1_t (*C++ struct*), 579
nvmlVgpuProcessesUtilizationInfo_v1_t::lastSeenTime (*C++ member*), 579
nvmlVgpuProcessesUtilizationInfo_v1_t::version (*C++ member*), 579
nvmlVgpuProcessesUtilizationInfo_v1_t::vgpuProcess (*C++ member*), 579
nvmlVgpuProcessesUtilizationInfo_v1_t::vgpuProcUtil (*C++ member*), 580
nvmlVgpuProcessUtilizationInfo_v1_t (*C++ struct*), 578

nvmlVgpuProcessUtilizationInfo_v1_t::decUtil (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::encUtil (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::jpgUtil (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::memUtil (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::ofaUtil (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::pid (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::processName (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::smUtil (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::timeStamp (*C++ member*), 578
nvmlVgpuProcessUtilizationInfo_v1_t::vgpuInstance (*C++ member*), 578
nvmlVgpuProcessUtilizationSample_t (*C++ struct*), 579
nvmlVgpuProcessUtilizationSample_t::decUtil (*C++ member*), 579
nvmlVgpuProcessUtilizationSample_t::encUtil (*C++ member*), 579
nvmlVgpuProcessUtilizationSample_t::memUtil (*C++ member*), 579
nvmlVgpuProcessUtilizationSample_t::pid (*C++ member*), 579
nvmlVgpuProcessUtilizationSample_t::processName (*C++ member*), 579
nvmlVgpuProcessUtilizationSample_t::smUtil (*C++ member*), 579
nvmlVgpuProcessUtilizationSample_t::timeStamp (*C++ member*), 579
nvmlVgpuProcessUtilizationSample_t::vgpuInstance (*C++ member*), 579
nvmlVgpuRuntimeState_t (*C++ type*), 449
nvmlVgpuRuntimeState_v1 (*C macro*), 447
nvmlVgpuRuntimeState_v1_t (*C++ struct*), 580
nvmlVgpuRuntimeState_v1_t::size (*C++ member*), 580
nvmlVgpuRuntimeState_v1_t::version (*C++ member*), 580
nvmlVgpuSchedulerCapabilities_t (*C++ struct*), 580
nvmlVgpuSchedulerCapabilities_t::isArrModeSupported (*C++ member*), 580
nvmlVgpuSchedulerCapabilities_t::maxAvgFactorForARR (*C++ member*), 580
nvmlVgpuSchedulerCapabilities_t::maxFrequencyForARR (*C++ member*), 580
nvmlVgpuSchedulerCapabilities_t::maxTimeslice (*C++ member*), 580
nvmlVgpuSchedulerCapabilities_t::minAvgFactorForARR (*C++ member*), 581
nvmlVgpuSchedulerCapabilities_t::minFrequencyForARR (*C++ member*), 580
nvmlVgpuSchedulerCapabilities_t::minTimeslice (*C++ member*), 580
nvmlVgpuSchedulerCapabilities_t::supportedSchedulers (*C++ member*), 580
nvmlVgpuSchedulerGetState_t (*C++ struct*), 581
nvmlVgpuSchedulerGetState_t::arrMode (*C++ member*), 581
nvmlVgpuSchedulerGetState_t::schedulerParams (*C++ member*), 581
nvmlVgpuSchedulerGetState_t::schedulerPolicy (*C++ member*), 581
nvmlVgpuSchedulerLog_t (*C++ struct*), 583
nvmlVgpuSchedulerLog_t::arrMode (*C++ member*), 584
nvmlVgpuSchedulerLog_t::engineId (*C++ member*), 584
nvmlVgpuSchedulerLog_t::entriesCount (*C++ member*), 584
nvmlVgpuSchedulerLog_t::logEntries (*C++ member*), 584
nvmlVgpuSchedulerLog_t::schedulerParams (*C++ member*), 584
nvmlVgpuSchedulerLog_t::schedulerPolicy (*C++ member*), 584
nvmlVgpuSchedulerLogEntry_t (*C++ struct*), 581
nvmlVgpuSchedulerLogEntry_t::cumulativePreemptionTime (*C++ member*), 581
nvmlVgpuSchedulerLogEntry_t::swRunlistId (*C++ member*), 581
nvmlVgpuSchedulerLogEntry_t::targetTimeSlice (*C++ member*), 581
nvmlVgpuSchedulerLogEntry_t::timeRun (*C++ member*), 581
nvmlVgpuSchedulerLogEntry_t::timeRunTotal (*C++ member*), 581
nvmlVgpuSchedulerLogEntry_t::timestamp (*C++ member*), 581
nvmlVgpuSchedulerLogEntry_v2_t (*C++ struct*), 582
nvmlVgpuSchedulerLogEntry_v2_t::cumulativePreemptionTime (*C++ member*), 582
nvmlVgpuSchedulerLogEntry_v2_t::swRunlistId (*C++ member*), 582
nvmlVgpuSchedulerLogEntry_v2_t::targetTimeSlice (*C++ member*), 582
nvmlVgpuSchedulerLogEntry_v2_t::timeRun (*C++ member*), 582

nvmlVgpuSchedulerLogEntry_v2_t::timeRunTotal (*C++ member*), 582
nvmlVgpuSchedulerLogEntry_v2_t::timestamp (*C++ member*), 582
nvmlVgpuSchedulerLogEntry_v2_t::weight (*C++ member*), 582
nvmlVgpuSchedulerLogInfo_t (*C++ type*), 449
nvmlVgpuSchedulerLogInfo_v1 (*C macro*), 447
nvmlVgpuSchedulerLogInfo_v1_t (*C++ struct*), 582
nvmlVgpuSchedulerLogInfo_v1_t::arrMode (*C++ member*), 582
nvmlVgpuSchedulerLogInfo_v1_t::engineId (*C++ member*), 582
nvmlVgpuSchedulerLogInfo_v1_t::entriesCount (*C++ member*), 583
nvmlVgpuSchedulerLogInfo_v1_t::logEntries (*C++ member*), 583
nvmlVgpuSchedulerLogInfo_v1_t::schedulerParams (*C++ member*), 583
nvmlVgpuSchedulerLogInfo_v1_t::schedulerPolicy (*C++ member*), 582
nvmlVgpuSchedulerLogInfo_v1_t::version (*C++ member*), 582
nvmlVgpuSchedulerLogInfo_v2_t (*C++ struct*), 583
nvmlVgpuSchedulerLogInfo_v2_t::avgFactor (*C++ member*), 583
nvmlVgpuSchedulerLogInfo_v2_t::engineId (*C++ member*), 583
nvmlVgpuSchedulerLogInfo_v2_t::entriesCount (*C++ member*), 583
nvmlVgpuSchedulerLogInfo_v2_t::logEntries (*C++ member*), 583
nvmlVgpuSchedulerLogInfo_v2_t::schedulerPolicy (*C++ member*), 583
nvmlVgpuSchedulerLogInfo_v2_t::timeslice (*C++ member*), 583
nvmlVgpuSchedulerParams_t (*C++ union*), 600
nvmlVgpuSchedulerParams_t::avgFactor (*C++ member*), 600
nvmlVgpuSchedulerParams_t::timeslice (*C++ member*), 600
nvmlVgpuSchedulerParams_t::vgpuSchedData (*C++ member*), 600
nvmlVgpuSchedulerParams_t::vgpuSchedDataWithARR (*C++ member*), 600
nvmlVgpuSchedulerSetParams_t (*C++ union*), 600
nvmlVgpuSchedulerSetParams_t::avgFactor (*C++ member*), 600
nvmlVgpuSchedulerSetParams_t::frequency (*C++ member*), 600
nvmlVgpuSchedulerSetParams_t::timeslice (*C++ member*), 601
nvmlVgpuSchedulerSetParams_t::vgpuSchedData (*C++ member*), 601
nvmlVgpuSchedulerSetParams_t::vgpuSchedDataWithARR (*C++ member*), 601
nvmlVgpuSchedulerSetState_t (*C++ struct*), 584
nvmlVgpuSchedulerSetState_t::enableARRMode (*C++ member*), 584
nvmlVgpuSchedulerSetState_t::schedulerParams (*C++ member*), 584
nvmlVgpuSchedulerSetState_t::schedulerPolicy (*C++ member*), 584
nvmlVgpuSchedulerState_t (*C++ type*), 449
nvmlVgpuSchedulerState_v1 (*C macro*), 447
nvmlVgpuSchedulerState_v1_t (*C++ struct*), 585
nvmlVgpuSchedulerState_v1_t::enableARRMode (*C++ member*), 586
nvmlVgpuSchedulerState_v1_t::engineId (*C++ member*), 585
nvmlVgpuSchedulerState_v1_t::schedulerParams (*C++ member*), 586
nvmlVgpuSchedulerState_v1_t::schedulerPolicy (*C++ member*), 586
nvmlVgpuSchedulerState_v1_t::version (*C++ member*), 585
nvmlVgpuSchedulerState_v2_t (*C++ struct*), 586
nvmlVgpuSchedulerState_v2_t::avgFactor (*C++ member*), 586
nvmlVgpuSchedulerState_v2_t::engineId (*C++ member*), 586
nvmlVgpuSchedulerState_v2_t::frequency (*C++ member*), 586
nvmlVgpuSchedulerState_v2_t::schedulerPolicy (*C++ member*), 586
nvmlVgpuSchedulerStateInfo_t (*C++ type*), 449
nvmlVgpuSchedulerStateInfo_v1 (*C macro*), 447
nvmlVgpuSchedulerStateInfo_v1_t (*C++ struct*), 584
nvmlVgpuSchedulerStateInfo_v1_t::arrMode (*C++ member*), 585
nvmlVgpuSchedulerStateInfo_v1_t::engineId (*C++ member*), 584
nvmlVgpuSchedulerStateInfo_v1_t::schedulerParams (*C++ member*), 585
nvmlVgpuSchedulerStateInfo_v1_t::schedulerPolicy (*C++ member*), 585
nvmlVgpuSchedulerStateInfo_v1_t::version (*C++ member*), 584
nvmlVgpuSchedulerStateInfo_v2_t (*C++*

struct), 585
nvmlVgpuSchedulerStateInfo_v2_t::avgFactor (*C++ member*), 585
nvmlVgpuSchedulerStateInfo_v2_t::engineId (*C++ member*), 585
nvmlVgpuSchedulerStateInfo_v2_t::schedulerPolicy (*C++ member*), 585
nvmlVgpuSchedulerStateInfo_v2_t::timeslice (*C++ member*), 585
nvmlVgpuTypeBar1Info_t (*C++ type*), 449
nvmlVgpuTypeBar1Info_v1 (*C macro*), 448
nvmlVgpuTypeBar1Info_v1_t (*C++ struct*), 586
nvmlVgpuTypeBar1Info_v1_t::bar1Size (*C++ member*), 586
nvmlVgpuTypeBar1Info_v1_t::version (*C++ member*), 586
nvmlVgpuTypeGetBAR1Info (*C++ function*), 474
nvmlVgpuTypeGetCapabilities (*C++ function*), 474
nvmlVgpuTypeGetClass (*C++ function*), 475
nvmlVgpuTypeGetDeviceID (*C++ function*), 475
nvmlVgpuTypeGetFbReservation (*C++ function*), 434
nvmlVgpuTypeGetFramebufferSize (*C++ function*), 476
nvmlVgpuTypeGetFrameRateLimit (*C++ function*), 475
nvmlVgpuTypeGetGpuInstanceProfileId (*C++ function*), 476
nvmlVgpuTypeGetGspHeapSize (*C++ function*), 434
nvmlVgpuTypeGetLicense (*C++ function*), 477
nvmlVgpuTypeGetMaxInstances (*C++ function*), 477
nvmlVgpuTypeGetMaxInstancesPerGpuInstance (*C++ function*), 478
nvmlVgpuTypeGetMaxInstancesPerVm (*C++ function*), 478
nvmlVgpuTypeGetName (*C++ function*), 479
nvmlVgpuTypeGetNumDisplayHeads (*C++ function*), 479
nvmlVgpuTypeGetResolution (*C++ function*), 479
nvmlVgpuTypeId_t (*C++ type*), 449
nvmlVgpuTypeIdInfo_t (*C++ type*), 449
nvmlVgpuTypeIdInfo_v1 (*C macro*), 448
nvmlVgpuTypeIdInfo_v1_t (*C++ struct*), 587
nvmlVgpuTypeIdInfo_v1_t::version (*C++ member*), 587
nvmlVgpuTypeIdInfo_v1_t::vgpuCount (*C++ member*), 587
nvmlVgpuTypeIdInfo_v1_t::vgpuTypeIds (*C++ member*), 587
nvmlVgpuTypeMaxInstance_t (*C++ type*), 449
nvmlVgpuTypeMaxInstance_v1 (*C macro*), 448
nvmlVgpuTypeMaxInstance_v1_t (*C++ struct*), 587
nvmlVgpuTypeMaxInstance_v1_t::maxInstancePerGI (*C++ member*), 587
nvmlVgpuTypeMaxInstance_v1_t::version (*C++ member*), 587
nvmlVgpuTypeMaxInstance_v1_t::vgpuTypeId (*C++ member*), 587
nvmlVgpuVersion_t (*C++ struct*), 587
nvmlVgpuVersion_t::maxVersion (*C++ member*), 587
nvmlVgpuVersion_t::minVersion (*C++ member*), 587
nvmlVgpuVmCompatibility_t (*C++ enum*), 482
nvmlVgpuVmCompatibility_t::NVML_VGPU_VM_COMPATIBILI (*C++ enumerator*), 482
nvmlVgpuVmCompatibility_t::NVML_VGPU_VM_COMPATIBILI (*C++ enumerator*), 482
nvmlVgpuVmCompatibility_t::NVML_VGPU_VM_COMPATIBILI (*C++ enumerator*), 482
nvmlVgpuVmCompatibility_t::NVML_VGPU_VM_COMPATIBILI (*C++ enumerator*), 482
nvmlVgpuVmCompatibility_t::NVML_VGPU_VM_COMPATIBILI (*C++ enumerator*), 482
nvmlVgpuVmIdType_t (*C++ enum*), 441
nvmlVgpuVmIdType_t::NVML_VGPU_VM_ID_DOMAIN_ID (*C++ enumerator*), 441
nvmlVgpuVmIdType_t::NVML_VGPU_VM_ID_UUID (*C++ enumerator*), 441
nvmlViolationTime_t (*C++ struct*), 588
nvmlViolationTime_t::referenceTime (*C++ member*), 588
nvmlViolationTime_t::violationTime (*C++ member*), 588
nvmlWorkloadPowerProfileCurrentProfiles_t (*C++ type*), 410
nvmlWorkloadPowerProfileCurrentProfiles_v1 (*C macro*), 404
nvmlWorkloadPowerProfileCurrentProfiles_v1_t (*C++ struct*), 588
nvmlWorkloadPowerProfileCurrentProfiles_v1_t::enfo (*C++ member*), 588
nvmlWorkloadPowerProfileCurrentProfiles_v1_t::perfP (*C++ member*), 588
nvmlWorkloadPowerProfileCurrentProfiles_v1_t::reque (*C++ member*), 588
nvmlWorkloadPowerProfileCurrentProfiles_v1_t::versi (*C++ member*), 588
nvmlWorkloadPowerProfileInfo_t (*C++ type*), 410
nvmlWorkloadPowerProfileInfo_v1 (*C macro*), 404
nvmlWorkloadPowerProfileInfo_v1_t (*C++*

struct), 588
nvmlWorkloadPowerProfileInfo_v1_t::conflictingMask
(*C++ member*), 589
nvmlWorkloadPowerProfileInfo_v1_t::priority
(*C++ member*), 588
nvmlWorkloadPowerProfileInfo_v1_t::profileId
(*C++ member*), 588
nvmlWorkloadPowerProfileInfo_v1_t::version
(*C++ member*), 588
nvmlWorkloadPowerProfileProfilesInfo_t
(*C++ type*), 410
nvmlWorkloadPowerProfileProfilesInfo_v1
(*C macro*), 405
nvmlWorkloadPowerProfileProfilesInfo_v1_t
(*C++ struct*), 589
nvmlWorkloadPowerProfileProfilesInfo_v1_t::perfProfile
(*C++ member*), 589
nvmlWorkloadPowerProfileProfilesInfo_v1_t::perfProfilesMask
(*C++ member*), 589
nvmlWorkloadPowerProfileProfilesInfo_v1_t::version
(*C++ member*), 589
nvmlWorkloadPowerProfileRequestedProfiles_t
(*C++ type*), 410
nvmlWorkloadPowerProfileRequestedProfiles_v1
(*C macro*), 405
nvmlWorkloadPowerProfileRequestedProfiles_v1_t
(*C++ struct*), 589
nvmlWorkloadPowerProfileRequestedProfiles_v1_t::requestedProfilesMask
(*C++ member*), 589
nvmlWorkloadPowerProfileRequestedProfiles_v1_t::version
(*C++ member*), 589
nvmlWorkloadPowerProfileUpdateProfiles_v1
(*C macro*), 405
nvmlWorkloadPowerProfileUpdateProfiles_v1_t
(*C++ struct*), 589
nvmlWorkloadPowerProfileUpdateProfiles_v1_t::operation
(*C++ member*), 589
nvmlWorkloadPowerProfileUpdateProfiles_v1_t::updateProfilesMask
(*C++ member*), 589