# TABLE OF CONTENTS

Chapter 1. NVML API Reference.............................................................................. 1  
Chapter 2. Known Issues....................................................................................... 3  
Chapter 3. Change Log..........................................................................................4  
Chapter 4. Modules.............................................................................................. 9  
  4.1. Device Structs............................................................................................10  
    nvmlPciInfo_t................................................................................................10  
    nvmlEccErrorCounts_t......................................................................................10  
    nvmlUtilization_t........................................................................................... 10  
    nvmlMemory_t.............................................................................................. 10  
    nvmlBAR1Memory_t........................................................................................ 10  
    nvmlProcessInfo_t.......................................................................................... 10  
    nvmlNVLinkUtilizationControl_t..........................................................................10  
    nvmlBridgeChipInfo_t......................................................................................10  
    nvmlBridgeChipHierarchy_t...............................................................................10  
    nvmlValue_t................................................................................................. 10  
    nvmlSample_t............................................................................................... 10  
    nvmlViolationTime_t....................................................................................... 10  
    nvmlBridgeChipType_t....................................................................................10  
    nvmlNVLinkUtilizationCountUnits_t......................................................................11  
    nvmlNVLinkUtilizationCountPktTypes_t................................................................11  
    nvmlNVLinkCapability_t....................................................................................11  
    nvmlNVLinkErrorCounter_t................................................................................12  
    nvmlGpuTopologyLevel_t.................................................................................. 12  
    nvmlSamplingType_t....................................................................................... 12  
    nvmlPcieUtilCounter_t.....................................................................................13  
    nvmlValueType_t............................................................................................13  
    nvmlPerfPolicyType_t...................................................................................... 13  
    NVML_VALUE_NOT_AVAILABLE......................................................................... 13  
    NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE............................................................... 14  
    NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE......................................................... 14  
    NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT............................................................. 14  
    NVML_DEVICE_PCI_BUS_ID_FMT......................................................................... 14  
    NVML_DEVICE_PCI_BUS_ID_FMT_ARGS.................................................................. 14  
    NVML_NVLINK_MAX_LINKS.................................................................................14  
    NVML_MAX_PHYSICAL_BRIDGE...........................................................................15  
  4.2. Device Enums............................................................................................15  
    nvmlEnableState_t......................................................................................... 15  
    nvmlBrandType_t........................................................................................... 15  
    nvmlTemperatureThresholds_t........................................................................... 15  
    nvmlTemperatureSensors_t............................................................................... 15  

www.nvidia.com  
NVML  
vR418 | ii
nvmlComputeMode_t.......................................................................................16
nvmlMemoryErrorType_t...................................................................................16
nvmlEccCounterType_t.....................................................................................17
nvmlClockType_t............................................................................................17
nvmlClockId_t............................................................................................. 17
nvmlDriverModel_t......................................................................................... 18
nvmlPstates_t............................................................................................... 18
nvmlGpuOperationMode_t................................................................................ 19
nvmlInforomObject_t...................................................................................... 19
nvmlReturn_t................................................................................................ 20
nvmlMemoryLocation_t.................................................................................... 21
nvmlPageRetirementCause_t........................................................................... 22
nvmlRestrictedAPI_t...................................................................................... 22
nvmlFlagDefault.............................................................................................22
nvmlFlagForce.............................................................................................22
nvmlEccBitType_t...........................................................................................22
NVML_SINGLE_BIT_ECC.................................................................................... 23
NVML_DOUBLE_BIT_ECC................................................................................... 23
4.3. GRID Enums.............................................................................................. 23
nvmlGpuVirtualizationMode_t......................................................................... 23
4.4. Field Value Enums.......................................................................................23
nvmlFieldValue_t........................................................................................... 24
NVML_FI_DEV_ECC_CURRENT............................................................................. 24
NVML_FI_DEV_ECC_PENDING.......................................................................... 24
NVML_FI_DEV_ECC_SBE_VOL_TOTAL................................................................... 24
NVML_FI_DEV_ECC_DBE_VOL_TOTAL................................................................... 24
NVML_FI_DEV_ECC_SBE_AGG_TOTAL.................................................................. 24
NVML_FI_DEV_ECC_DBE_AGG_TOTAL.................................................................. 24
NVML_FI_DEV_ECC_SBE_VOL_L1.......................................................................... 24
NVML_FI_DEV_ECC_DBE_VOL_L1......................................................................... 24
NVML_FI_DEV_ECC_SBE_VOL_L2.......................................................................... 24
NVML_FI_DEV_ECC_DBE_VOL_L2......................................................................... 24
NVML_FI_DEV_ECC_SBE_VOL_DEV........................................................................ 25
NVML_FI_DEV_ECC_DBE_VOL_DEV....................................................................... 25
NVML_FI_DEV_ECC_SBE_VOL_REG........................................................................ 25
NVML_FI_DEV_ECC_DBE_VOL_REG....................................................................... 25
NVML_FI_DEV_ECC_SBE_VOL_TEX........................................................................ 25
NVML_FI_DEV_ECC_DBE_VOL_TEX....................................................................... 25
NVML_FI_DEV_ECC_DBE_VOL_CBU....................................................................... 25
NVML_FI_DEV_ECC_SBE_AGG_L1......................................................................... 25
NVML_FI_DEV_ECC_DBE_AGG_L1......................................................................... 25
NVML_FI_DEV_ECC_SBE_AGG_L2......................................................................... 25
NVML_FI_DEV_ECC_DBE_AGG_L2......................................................................... 25
| NVML_FI_DEV_ECC_SBE_AGG_DEV | 26 |
| NVML_FI_DEV_ECC_DBE_AGG_DEV | 26 |
| NVML_FI_DEV_ECC_SBE_AGG_REG | 26 |
| NVML_FI_DEV_ECC_DBE_AGG_REG | 26 |
| NVML_FI_DEV_ECC_SBE_AGG_TEX | 26 |
| NVML_FI_DEV_ECC_DBE_AGG_TEX | 26 |
| NVML_FI_DEV_ECC_DBE_AGG_CBU | 26 |
| NVML_FI_DEV_RETIRED_SBE | 26 |
| NVML_FI_DEV_RETIRED_DBE | 26 |
| NVML_FI_DEV_RETIRED_PENDING | 26 |
| NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L0 | 26 |
| NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L1 | 27 |
| NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L2 | 27 |
| NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L3 | 27 |
| NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L4 | 27 |
| NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L5 | 27 |
| NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_TOTAL | 28 |
| NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L0 | 27 |
| NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L1 | 27 |
| NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L2 | 28 |
| NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L3 | 28 |
| NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L4 | 28 |
| NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L5 | 28 |
| NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_TOTAL | 28 |
| NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L0 | 28 |
| NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L1 | 28 |
| NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L2 | 28 |
| NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L3 | 29 |
| NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L4 | 29 |
| NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L5 | 29 |
| NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_TOTAL | 29 |
| NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L0 | 29 |
| NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L1 | 29 |
| NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L2 | 29 |
| NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L3 | 29 |
| NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L4 | 30 |
| NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L5 | 30 |
| NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_TOTAL | 30 |
| NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L0 | 30 |
| NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L1 | 30 |
| NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L2 | 30 |
| NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L3 | 30 |
| NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L4 | 30 |
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L5............................................................. 30
NVML_FI_DEV_NVLINK_BANDWIDTH_C0_TOTAL.................................................. 30
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L0.......................................................... 31
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L1.......................................................... 31
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L2.......................................................... 31
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L3.......................................................... 31
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L4.......................................................... 31
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L5.......................................................... 31
NVML_FI_DEV_NVLINK_BANDWIDTH_C1_TOTAL.................................................. 31
NVML_FI_DEV_PERF_POLICY_POWER.................................................................... 31
NVML_FI_DEV_PERF_POLICY_THERMAL.................................................................. 31
NVML_FI_DEV_PERF_POLICY_SYNC_BOOST............................................................. 31
NVML_FI_DEV_PERF_POLICY_BOARD_LIMIT............................................................ 31
NVML_FI_DEV_PERF_POLICY_LOW_UTILIZATION..................................................... 32
NVML_FI_DEV_PERF_POLICY_RELIABILITY............................................................ 32
NVML_FI_DEV_PERF_POLICY_TOTAL_APP_CLOCKS................................................. 32
NVML_FI_DEV_PERF_POLICY_TOTAL_BASE_CLOCKS............................................... 32
NVML_FI_DEV_MEMORY_TEMP............................................................................. 32
NVML_FI_DEV_TOTAL_ENERGY_CONSUMPTION....................................................... 32
NVML_FI_DEV_NVLINK_SPEED_MBPS_L0............................................................... 32
NVML_FI_DEV_NVLINK_SPEED_MBPS_L1............................................................... 32
NVML_FI_DEV_NVLINK_SPEED_MBPS_L2............................................................... 32
NVML_FI_DEV_NVLINK_SPEED_MBPS_L3............................................................... 32
NVML_FI_DEV_NVLINK_SPEED_MBPS_L4............................................................... 33
NVML_FI_DEV_NVLINK_SPEED_MBPS_L5............................................................... 33
NVML_FI_DEV_NVLINK_SPEED_MBPS_COMMON...................................................... 33
NVML_FI_DEV_NVLINK_LINK_COUNT..................................................................... 33
NVML_FI_DEV_RETIRED_PENDING_SBE.................................................................. 33
NVML_FI_DEV_RETIRED_PENDING_DBE.................................................................. 33
NVML_FI_DEV_PCIE_REPLAY_COUNTER.................................................................. 33
NVML_FI_DEV_PCIE_REPLAY_ROLLOVER_COUNTER.................................................... 33
NVML_FI_MAX................................................................................................ 33
4.5. Unit Structs............................................................................................... 33
nvmlHwbcEntry_t........................................................................................... 34
nvmlLedState_t............................................................................................. 34
nvmlUnitInfo_t.............................................................................................. 34
nvmlPSUInfo_t............................................................................................... 34
nvmlUnitFanInfo_t.......................................................................................... 34
nvmlUnitFanSpeeds_t...................................................................................... 34
nvmlFanState_t............................................................................................. 34
nvmlLedColor_t............................................................................................. 34
4.6. Accounting Statistics.................................................................................. 34
nvmlAccountingStats_t..................................................................................... 35
4.7. Vgpu Constants........................................40
   NVML_GRID_LICENSE_BUFFER_SIZE..................................................40
   NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION................................40
4.8. Vgpu Enum ..................................................40
   nvmlVgpuVmIdType_t.........................................................................40
   nvmlVgpuGuestInfoState_t................................................................40
   nvmlGridLicenseFeatureCode_t..........................................................40
4.9. Vgpu Structs .................................................41
   nvmlVgpuInstanceUtilizationSample_t...............................................41
   nvmlVgpuProcessUtilizationSample_t...............................................41
   nvmlProcessUtilizationSample_t......................................................41
   nvmlGridLicensableFeature_t............................................................41
   nvmlGridLicensableFeatures_t.............................................................41
4.10. Encoder Structs..............................................41
   nvmlEncoderSessionInfo_t................................................................41
   nvmlEncoderType_t..........................................................................41
4.11. Frame Buffer Capture Structures.........................41
   nvmlFBCStats_t..............................................................................42
   nvmlFBCSessionInfo_t....................................................................42
   nvmlFBCSessionType_t....................................................................42
   NVML_NVFBC_SESSION_FLAG_DIFFMAP_ENABLED..................................42
   NVML_NVFBC_SESSION_FLAG_CLASSIFICATIONMAP_ENABLED..................42
   NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_NO_WAIT..................42
   NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_INFINITE................43
   NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_TIMEOUT................43
4.12. definitions related to the drain state.......................43
   nvmlDetachGpuState_t....................................................................43
   nvmlPcieLinkState_t......................................................................43
4.13. Initialization and Cleanup ..................................43
   nvmlInit.................................................................43
   nvmlInitWithFlags..............................................................44
   nvmlShutdown.................................................................45
   NVML_INIT_FLAG_NO_GPUS......................................................45
   NVML_INIT_FLAG_NO_ATTACH..................................................45
4.14. Error reporting...............................................45
   nvmlErrorString........................................................................46
4.15. Constants......................................................46

nvmlDeviceGetAccountingMode.................................................................35
nvmlDeviceGetAccountingStats.................................................................35
nvmlDeviceGetAccountingPids.................................................................36
nvmlDeviceGetAccountingBufferSize.......................................................37
nvmlDeviceSetAccountingMode.................................................................38
nvmlDeviceClearAccountingPids...............................................................39
4.16. System Queries......................................................................................... 47
   nvmlSystemGetDriverVersion...................................................................... 47
   nvmlSystemGetNVMLVersion...................................................................... 48
   nvmlSystemGetCudaDriverVersion............................................................ 48
   nvmlSystemGetCudaDriverVersion_v2...................................................... 49
   nvmlSystemGetProcessName.................................................................... 49
   NVML_CUDA_DRIVER_VERSION_MAJOR.......................................................... 50

4.17. Unit Queries............................................................................................ 50
   nvmlUnitGetCount.................................................................................. 50
   nvmlUnitGetHandleByIndex......................................................................... 51
   nvmlUnitGetUnitInfo............................................................................... 51
   nvmlUnitGetLedState............................................................................... 52
   nvmlUnitGetPsuInfo............................................................................... 53
   nvmlUnitGetTemperature......................................................................... 53
   nvmlUnitGetFanSpeedInfo....................................................................... 54
   nvmlUnitGetDevices............................................................................... 54
   nvmlSystemGetHicVersion...................................................................... 55

4.18. Device Queries......................................................................................... 56
   nvmlDeviceGetCount.............................................................................. 56
   nvmlDeviceGetHandleByIndex.................................................................... 57
   nvmlDeviceGetHandleBySerial................................................................... 58
   nvmlDeviceGetHandleByUUID.................................................................... 59
   nvmlDeviceGetHandleByPciBusId............................................................ 60
   nvmlDeviceGetName............................................................................... 61
   nvmlDeviceGetBrand............................................................................. 62
   nvmlDeviceGetIndex............................................................................... 62
   nvmlDeviceGetSerial............................................................................. 63
   nvmlDeviceGetCpuAffinity...................................................................... 64
   nvmlDeviceSetCpuAffinity...................................................................... 65
   nvmlDeviceClearCpuAffinity................................................................... 65
   nvmlDeviceGetTopologyCommonAncestor............................................... 66
   nvmlDeviceGetTopologyNearestGpus....................................................... 66
   nvmlSystemGetTopologyGpuSet.............................................................. 67
   nvmlDeviceGetP2PStatus......................................................................... 68
   nvmlDeviceGetUUID.............................................................................. 69
nvmlDeviceGetMinorNumber.................................................................69
nvmlDeviceGetBoardPartNumber..........................................................70
nvmlDeviceGetInforomVersion...............................................................71
nvmlDeviceGetInforomImageVersion......................................................72
nvmlDeviceGetInforomConfigurationChecksum.......................................73
nvmlDeviceValidateInforom..................................................................73
nvmlDeviceGetDisplayMode.................................................................74
nvmlDeviceGetDisplayActive...............................................................75
nvmlDeviceGetPersistenceMode............................................................75
nvmlDeviceGetPciInfo...........................................................................76
nvmlDeviceGetMaxPcieLinkGeneration...................................................77
nvmlDeviceGetMaxPcieLinkWidth............................................................77
nvmlDeviceGetCurrPcieLinkGeneration...................................................78
nvmlDeviceGetCurrPcieLinkWidth..........................................................79
nvmlDeviceGetPcieThroughput..............................................................79
nvmlDeviceGetPcieReplayCounter..........................................................80
nvmlDeviceGetClockInfo......................................................................81
nvmlDeviceGetMaxClockInfo..................................................................81
nvmlDeviceGetApplicationsClock...........................................................82
nvmlDeviceGetDefaultApplicationsClock...............................................83
nvmlDeviceResetApplicationsClocks.......................................................84
nvmlDeviceGetClock............................................................................85
nvmlDeviceGetMaxCustomerBoostClock..................................................85
nvmlDeviceGetSupportedMemoryClocks...................................................86
nvmlDeviceGetSupportedGraphicsClocks..................................................87
nvmlDeviceGetAutoBoostedClocksEnabled...............................................88
nvmlDeviceSetAutoBoostedClocksEnabled...............................................89
nvmlDeviceSetDefaultAutoBoostedClocksEnabled......................................90
nvmlDeviceGetFanSpeed.......................................................................91
nvmlDeviceGetFanSpeed_v2....................................................................91
nvmlDeviceGetTemperature...................................................................92
nvmlDeviceGetTemperatureThreshold....................................................93
nvmlDeviceGetPerformanceState............................................................94
nvmlDeviceGetCurrentClocksThrottleReasons..........................................94
nvmlDeviceGetSupportedClocksThrottleReasons........................................95
nvmlDeviceGetPowerState.....................................................................96
nvmlDeviceGetPowerManagementMode..................................................97
nvmlDeviceGetPowerManagementLimit..................................................98
nvmlDeviceGetPowerManagementLimitConstraints....................................98
nvmlDeviceGetPowerManagementDefaultLimit.......................................99
nvmlDeviceGetPowerUsage...................................................................100
nvmlDeviceGetTotalEnergyConsumption...............................................101
nvmlDeviceGetEnforcedPowerLimit.......................................................101
4.21. NvLink Methods

4.22. Event Handling Methods

4.22.1. Event Types

4.23. Drain states

4.24. Field Value Queries

4.25. Grid Queries

4.26. Grid Commands

4.27. vGPU Management
nvmlDeviceGetSupportedVgpus................................................................. 160
nvmlDeviceGetCreatableVgpus.................................................................. 161
nvmlVgpuTypeGetClass............................................................................. 162
nvmlVgpuTypeGetName.............................................................................. 163
nvmlVgpuTypeGetDeviceID....................................................................... 164
nvmlVgpuTypeGetFramebufferSize............................................................. 164
nvmlVgpuTypeGetNumDisplayHeads........................................................... 165
nvmlVgpuTypeGetResolution..................................................................... 165
nvmlVgpuTypeGetLicense.......................................................................... 166
nvmlVgpuTypeGetFrameRateLimit............................................................... 167
nvmlVgpuTypeGetMaxInstances.................................................................. 168
nvmlVgpuTypeGetMaxInstancesPerVm......................................................... 168
nvmlDeviceGetActiveVgpus..................................................................... 169
nvmlVgpuInstanceGetVmID....................................................................... 170
nvmlVgpuInstanceGetUUID....................................................................... 171
nvmlVgpuInstanceGetVmDriverVersion....................................................... 171
nvmlVgpuInstanceGetFbUsage................................................................... 172
nvmlVgpuInstanceGetLicenseStatus........................................................... 173
nvmlVgpuInstanceGetType........................................................................ 174
nvmlVgpuInstanceGetFrameRateLimit......................................................... 174
nvmlVgpuInstanceGetEncoderCapacity...................................................... 175
nvmlVgpuInstanceSetEncoderCapacity....................................................... 176
nvmlDeviceGetVgpuUtilization................................................................. 176
nvmlDeviceGetVgpuProcessUtilization....................................................... 178
nvmlDeviceGetGridLicensableFeatures...................................................... 179
nvmlVgpuInstanceGetEncoderStats............................................................ 180
nvmlVgpuInstanceGetEncoderSessions...................................................... 181
nvmlVgpuInstanceGetFBCStats.................................................................. 182
nvmlVgpuInstanceGetFBCSessions............................................................ 182
nvmlDeviceGetProcessUtilization............................................................. 183
nvmlVgpuInstanceGetAccountingMode....................................................... 185
nvmlVgpuInstanceGetAccountingPids........................................................ 185
nvmlVgpuInstanceGetAccountingStats....................................................... 186
4.28. vGPU Migration............................................................................... 187
nvmlVgpuVersion_t.................................................................................. 188
nvmlVgpuMetadata_t.............................................................................. 188
nvmlVgpuPgpuMetadata_t........................................................................ 188
nvmlVgpuPgpuCompatibility_t.................................................................. 188
nvmlVgpuVmCompatibility_t.................................................................... 188
nvmlVgpuPgpuCompatibilityLimitCode_t.................................................. 188
nvmlVgpuInstanceGetMetadata............................................................... 189
nvmlDeviceGetVgpuMetadata................................................................. 190
nvmlGetVgpuCompatibility................................................................. 190
nvmlGetVgpuVersion...................................................................................... 191
nvmlSetVgpuVersion...................................................................................... 192

4.29. GPU Blacklist Queries........................................................................... 193
nvmlBlacklistDeviceInfo_t.............................................................................. 193
nvmlGetBlacklistDeviceCount........................................................................... 193
nvmlGetBlacklistDeviceInfoByIndex................................................................. 194

4.30. NvmlClocksThrottleReasons...................................................................... 194
nvmlClocksThrottleReasonGpuIdle.................................................................... 195
nvmlClocksThrottleReasonApplicationsClocksSetting........................................ 195
nvmlClocksThrottleReasonUserDefinedClocks................................................... 195
nvmlClocksThrottleReasonSwPowerCap............................................................. 195
nvmlClocksThrottleReasonHwSlowdown................................................................ 195
nvmlClocksThrottleReasonSyncBoost................................................................. 196
nvmlClocksThrottleReasonSwThermalSlowdown................................................ 196
nvmlClocksThrottleReasonHwThermalSlowdown............................................... 196
nvmlClocksThrottleReasonHwPowerBrakeSlowdown.......................................... 197
nvmlClocksThrottleReasonDisplayClockSetting................................................. 197
nvmlClocksThrottleReasonNone......................................................................... 197
nvmlClocksThrottleReasonAll............................................................................ 197

Chapter 5. Data Structures.............................................................................. 199
nvmlAccountingStats_t...................................................................................... 200
gpuUtilization................................................................................................... 200
memoryUtilization............................................................................................. 200
maxMemoryUsage............................................................................................... 200
time .................................................................................................................... 200
startTime........................................................................................................... 201
isRunning........................................................................................................... 201
reserved............................................................................................................ 201
nvmlBAR1Memory_t.......................................................................................... 201
bar1Total.......................................................................................................... 201
bar1Free........................................................................................................... 201
bar1Used.......................................................................................................... 201
nvmlBlacklistDeviceInfo_t.................................................................................. 201
pciInfo.............................................................................................................. 201
uuid.................................................................................................................. 201
nvmlBridgeChipHierarchy_t................................................................................. 201
bridgeCount...................................................................................................... 202
bridgeChipInfo................................................................................................. 202
nvmlBridgeChipInfo_t........................................................................................ 202
type.................................................................................................................. 202
fwVersion......................................................................................................... 202
nvmlEccErrorCounts_t....................................................................................... 202
l1Cache.............................................................................................................. 203
l2Cache..................................................................................................... 203
deviceMemory............................................................................................. 203
registerFile.................................................................................................203
nvmlEncoderSessionInfo_t.................................................................................. 203
sessionId.................................................................................................... 204
pid........................................................................................................... 204
vgpuInstance............................................................................................... 204
codecType.................................................................................................. 204
hResolution.................................................................................................204
vResolution................................................................................................. 204
averageFps................................................................................................. 204
averageLatency............................................................................................204
nvmlEventData_t............................................................................................. 204
device....................................................................................................... 205
eventType.................................................................................................. 205
eventData.................................................................................................. 205
nvmlFBCSessionInfo_t....................................................................................... 205
sessionId.................................................................................................... 206
pid........................................................................................................... 206
vgpuInstance............................................................................................... 206
displayOrdinal..............................................................................................206
sessionType.................................................................................................206
sessionFlags................................................................................................ 206
hMaxResolution............................................................................................ 206
vMaxResolution............................................................................................ 206
hResolution.................................................................................................206
vResolution................................................................................................. 206
averageFPS................................................................................................. 206
averageLatency............................................................................................206
nvmlFBCStats_t............................................................................................... 207
sessionsCount.............................................................................................. 207
averageFPS................................................................................................. 207
averageLatency............................................................................................207
nvmlFieldValue_t............................................................................................. 207
fieldId....................................................................................................... 208
unused...................................................................................................... 208
timestamp.................................................................................................. 208
latencyUsec................................................................................................ 208
valueType...................................................................................................208
nvmlReturn................................................................................................. 208
value........................................................................................................ 208
nvmlGridLicensableFeature_t.............................................................................. 208
featureCode................................................................................................. 209
featureState........................................................................................................209
nvmlGridLicensableFeatures_t............................................................................209
isGridLicenseSupported..................................................................................209
 licensableFeaturesCount.................................................................................209
gridLicensableFeatures..................................................................................209
nvmlHwbcEntry_t............................................................................................209
nvmlLedState_t...............................................................................................209
 cause..............................................................................................................210
color..............................................................................................................210
nvmlMemory_t................................................................................................210
total................................................................................................................210
free..................................................................................................................210
used..................................................................................................................210
nvmlNVLinkUtilizationControl_t............................................................................210
nvmlPciInfo_t.................................................................................................210
 busIdLegacy..................................................................................................211
domain............................................................................................................211
bus...................................................................................................................211
device..............................................................................................................211
pciDeviceId.....................................................................................................211
pciSubsystemId...............................................................................................211
busId................................................................................................................211
nvmlProcessInfo_t............................................................................................211
 pid...................................................................................................................211
usedGpuMemory.............................................................................................211
nvmlProcessUtilizationSample_t...........................................................................212
 pid...................................................................................................................212
timeStamp.......................................................................................................212
smUtil..............................................................................................................212
memUtil..........................................................................................................212
encUtil..........................................................................................................212
decUtil..........................................................................................................212
nvmlPSUInfo_t................................................................................................212
state................................................................................................................213
current.............................................................................................................213
voltage.............................................................................................................213
power..............................................................................................................213
nvmlSample_t..................................................................................................213
timeStamp.......................................................................................................213
sampleValue....................................................................................................213
nvmlUnitFanInfo_t............................................................................................213
 speed..............................................................................................................213
state..............................................................................................................213
Chapter 6. Data Fields ................................................................. 222
Chapter 7. Deprecated List ......................................................... 229
Chapter 1.
NVML API REFERENCE

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

API Documentation

Supported OS platforms:

- Linux: 32-bit and 64-bit

Supported products:

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

- NVIDIA Tesla Line: S1070, C1060, M1060 and all other previous generation Tesla-branded parts
- NVIDIA Quadro Line: All other current and previous generation Quadro-branded parts
- NVIDIA GeForce Line: All current and previous generation GeForce-branded parts

The NVML library can be found at: %ProgramW6432%"NVIDIA Corporation"\NVSMI on Windows, but will not be added to the path. To dynamically link to NVML, add this path to the PATH environmental variable. To dynamically load NVML, call LoadLibrary with this path.

On Linux the NVML library will be found on the standard library path. For 64-bit Linux, both the 32-bit and 64-bit NVML libraries will be installed.

The NVML API is divided into five categories:

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

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

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

‣ On Linux when X Server is running nvmlDeviceGetComputeRunningProcesses may return a nvmlProcessInfo_t::usedGpuMemory value that is larger than the actual value. This will be fixed in a future release.

‣ On Linux GPU Reset can't be triggered when there is pending GPU Operation Mode (GOM) change.

‣ On Linux GPU Reset may not successfully change pending ECC mode. A full reboot may be required to enable the mode change.

‣ nvmlAccountingStats supports only one process per GPU at a time (CUDA proxy server counts as one process).

‣ nvmlAccountingStats_t.time reports time and utilization values starting from cuInit till process termination. Next driver versions might change this behavior slightly and account process only from cuCtxCreate till cuCtxDestroy.

‣ On GPUs from Fermi family current P0 clocks (reported by nvmlDeviceGetClockInfo) can differ from max clocks by few MHz.
Chapter 3.
CHANGE LOG

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

Changes between v346 and v349
The following new functionality is exposed on NVIDIA display drivers version 349 Production or later

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

Changes between v340 and v346
The following new functionality is exposed on NVIDIA display drivers version 346 Production or later

- Added `nvmlDeviceGetGraphicsRunningProcesses` to get information about Graphics Processes running on a device.
- Added `nvmlDeviceGetPcieReplayCounter` to get PCI replay counters.
- Added `nvmlDeviceGetPcieThroughput` to get PCI utilization information.
- Discontinued Perl bindings support.
Changes between NVML v331 and v340

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

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

Changes between NVML v5.319 Update and v331

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

- Added `nvmlDeviceGetMinorNumber` to get the minor number for the device.
- Added `nvmlDeviceGetBAR1MemoryInfo` to get BAR1 total, available and used memory size.
- Added `nvmlDeviceGetBridgeChipInfo` to get the information related to bridge chip firmware.
- Added enforced power limit query API `nvmlDeviceGetEnforcedPowerLimit`
- Updated `nvmlEventSetWait` to return xid event data in case of xid error event.
Changes between NVML v5.319 RC and v5.319 Update

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

- Added `nvmlDeviceSetAPIRestriction` and `nvmlDeviceGetAPIRestriction`, with initial ability to toggle root-only requirement for `nvmlDeviceSetApplicationsClocks` and `nvmlDeviceResetApplicationsClocks`.

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

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

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

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

The following new functionality is exposed on NVIDIA display drivers version 304 Production or later.
Added \texttt{nvmlDeviceGetGpuOperationMode} and \texttt{nvmlDeviceSetGpuOperationMode}.

\textbf{Changes between NVML v3.295 and v4.304 RC}

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

\begin{itemize}
  \item Added \texttt{nvmlDeviceGetInforomConfigurationChecksum} and \texttt{nvmlDeviceValidateInforom}.
  \item Added \texttt{nvmlDeviceGetDisplayActive} and updated documentation to clarify how it differs from \texttt{nvmlDeviceGetDisplayMode}.
  \item Added new error return value for initialization failure due to kernel module not receiving interrupts.
  \item Added \texttt{nvmlDeviceSetApplicationsClocks}, \texttt{nvmlDeviceGetApplicationsClock}, \texttt{nvmlDeviceResetApplicationsClocks}.
  \item Added \texttt{nvmlDeviceGetSupportedMemoryClocks} and \texttt{nvmlDeviceGetSupportedGraphicsClocks}.
  \item Added \texttt{nvmlDeviceGetPowerManagementLimitConstraints}, \texttt{nvmlDeviceGetPowerManagementDefaultLimit} and \texttt{nvmlDeviceSetPowerManagementLimit}.
  \item Added \texttt{nvmlDeviceGetInforomImageVersion}.
  \item Expanded \texttt{nvmlDeviceGetUUID} to support all CUDA capable GPUs.
  \item Deprecated \texttt{nvmlDeviceGetDetailedEccErrors} in favor of \texttt{nvmlDeviceGetMemoryErrorCounter}.
  \item Added \texttt{NVML\_MEMORY\_LOCATION\_TEXTURE\_MEMORY} to support reporting of texture memory error counters.
  \item Added \texttt{nvmlDeviceGetCurrentClocksThrottleReasons} and \texttt{nvmlDeviceGetSupportedClocksThrottleReasons}.
  \item \texttt{NVML\_CLOCK\_SM} is now also reported on supported Kepler devices.
  \item Dropped support for GT200 based Tesla brand GPUs: C1060, M1060, S1070.
\end{itemize}

\textbf{Changes between NVML v2.285 and v3.295}

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

\begin{itemize}
  \item Deprecated \texttt{nvmlDeviceGetHandleBySerial} in favor of newly added \texttt{nvmlDeviceGetHandleByUUID}.
  \item Marked the input parameters of \texttt{nvmlDeviceGetHandleBySerial}, \texttt{nvmlDeviceGetHandleByUUID} and \texttt{nvmlDeviceGetHandleByPciBusId} as const.
  \item Added \texttt{nvmlDeviceOnSameBoard}.
  \item Added \texttt{nvmlConstants} defines.

Format change of nvmlDeviceGetUUID output to match the UUID standard. This function will return a different value.

nvmlDeviceGetDetailedEccErrors will report zero for unsupported ECC error counters when a subset of ECC error counters are supported.

Changes between NVML v1.0 and v2.285

The following new functionality is exposed on NVIDIA display drivers version 285 or later.

- Added possibility to query separately current and pending driver model with nvmlDeviceGetDriverModel.
- Added API nvmlDeviceGetVbiosVersion function to report VBIOS version.
- Added pciSubSystemId to nvmlPciInfo_t struct.
- Added API nvmlErrorString function to convert error code to string.
- Updated docs to indicate we support M2075 and C2075.
- Added API nvmlSystemGetHicVersion function to report HIC firmware version.
- Added NVML versioning support

  - Functions that changed API and/or size of structs have appended versioning suffix (e.g., nvmlDeviceGetPciInfo_v2). Appropriate C defines have been added that map old function names to the newer version of the function.

- Added support for concurrent library usage by multiple libraries.
- Added API nvmlDeviceGetMaxClockInfo function for reporting device's clock limits.
- Added new error code NVML_ERROR_DRIVER_NOT_LOADED used by nvmlInit.
- Extended nvmlPciInfo_t struct with new field: sub system id.
- Added NVML support on Windows guest account.
- Changed format of pciBusId string (to XXXX:XX:XX.X) of nvmlPciInfo_t.
- Parsing of busId in nvmlDeviceGetHandleByPciBusId is less restrictive. You can pass 0:2:0.0 or 0000:02:00 and other variations.
- Added API for events waiting for GPU events (Linux only) see docs of nvmlEvents.
- Added API nvmlDeviceGetComputeRunningProcesses and nvmlSystemGetProcessName functions for looking up currently running compute applications.
- Deprecated nvmlDeviceGetPowerState in favor of nvmlDeviceGetPerformanceState.
Chapter 4. 
MODULES

Here is a list of all modules:

- Device Structs
- Device Enums
- GRID Enums
- Field Value Enums
- Unit Structs
- Accounting Statistics
- Vgpu Constants
- Vgpu Enum
- Vgpu Structs
- Encoder Structs
- Frame Buffer Capture Structures
- definitions related to the drain state
- Initialization and Cleanup
- Error reporting
- Constants
- System Queries
- Unit Queries
- Device Queries
- Unit Commands
- Device Commands
- NvLink Methods
- Event Handling Methods
  - Event Types
  - Drain states
- Field Value Queries
- Grid Queries
4.1. Device Structs

```c
struct nvmlPciInfo_t
struct nvmlEccErrorCounts_t
struct nvmlUtilization_t
struct nvmlMemory_t
struct nvmlBAR1Memory_t
struct nvmlProcessInfo_t
struct nvmlNVLinkUtilizationControl_t
struct nvmlBridgeChipInfo_t
struct nvmlBridgeChipHierarchy_t
union nvmlValue_t
struct nvmlSample_t
struct nvmlViolationTime_t
enum nvmlBridgeChipType_t
```

Enum to represent type of bridge chip
Values

NVML_BRIDGE_CHIP_PLX = 0
NVML_BRIDGE_CHIP_BRO4 = 1

defined nvmlNvLinkUtilizationCountUnits_t

Enum to represent the NvLink utilization counter packet units

Values

NVML_NVLINK_COUNTER_UNIT_CYCLES = 0
NVML_NVLINK_COUNTER_UNIT_PACKETS = 1
NVML_NVLINK_COUNTER_UNIT_BYTES = 2
NVML_NVLINK_COUNTER_UNIT_COUNT

defined nvmlNvLinkUtilizationCountPktTypes_t

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

Values

NVML_NVLINK_COUNTER_PKTFILTER_NOP = 0x1
NVML_NVLINK_COUNTER_PKTFILTER_READ = 0x2
NVML_NVLINK_COUNTER_PKTFILTER_WRITE = 0x4
NVML_NVLINK_COUNTER_PKTFILTER_RATOM = 0x8
NVML_NVLINK_COUNTER_PKTFILTER_NRATOM = 0x10
NVML_NVLINK_COUNTER_PKTFILTER_FLUSH = 0x20
NVML_NVLINK_COUNTER_PKTFILTER_RESPDATA = 0x40
NVML_NVLINK_COUNTER_PKTFILTER_RESPNODATA = 0x80
NVML_NVLINK_COUNTER_PKTFILTER_ALL = 0xFF

defined nvmlNvLinkCapability_t

Enum to represent NvLink queryable capabilities

Values

NVML_NVLINK_CAP_P2P_SUPPORTED = 0
NVML_NVLINK_CAP_SYSMEM_ACCESS = 1
NVML_NVLINK_CAP_P2P_ATOMICS = 2
NVML_NVLINK_CAP_SYSMEM_ATOMICS = 3
NVML_NVLINK_CAP_SLI_BRIDGE = 4
NVML_NVLINK_CAP_VALID = 5
NVML_NVLINK_CAP_COUNT

enum nvmlNvLinkErrorCounter_t
Enum to represent NvLink queryable error counters

Values
NVML_NVLINK_ERROR_DL_REPLAY = 0
NVML_NVLINK_ERROR_DL_RECOVERY = 1
NVML_NVLINK_ERROR_DL_CRC_FLIT = 2
NVML_NVLINK_ERROR_DL_CRC_DATA = 3
NVML_NVLINK_ERROR_COUNT

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

Values
NVML_TOPOLOGY_INTERNAL = 0
NVML_TOPOLOGY_SINGLE = 10
NVML_TOPOLOGY_MULTIPLE = 20
NVML_TOPOLOGY_HOSTBRIDGE = 30
NVML_TOPOLOGY_NODE = 40
NVML_TOPOLOGY_SYSTEM = 50

enum nvmlSamplingType_t
Represents Type of Sampling Event

Values
NVML_TOTAL_POWER_SAMPLES = 0
   To represent total power drawn by GPU.
NVML_GPU_UTILIZATION_SAMPLES = 1
   To represent percent of time during which one or more kernels was executing on the
   GPU.
NVML_MEMORY_UTILIZATION_SAMPLES = 2
   To represent percent of time during which global (device) memory was being read or
   written.
NVML_ENC_UTILIZATION_SAMPLES = 3
   To represent percent of time during which NVENC remains busy.
NVML_DEC_UTILIZATION_SAMPLES = 4
   To represent percent of time during which NVDEC remains busy.
NVML_PROCESOR_CLK_SAMPLES = 5
To represent processor clock samples.
NVML_MEMORY_CLK_SAMPLES = 6
To represent memory clock samples.
NVML_SAMPLINGTYPE_COUNT

```
enum nvmlPcieUtilCounter_t
```

Represents the queryable PCIe utilization counters

**Values**

NVML_PCIE_UTIL_TX_BYTES = 0
NVML_PCIE_UTIL_RX_BYTES = 1
NVML_PCIE_UTIL_COUNT

```
enum nvmlValueType_t
```

Represents the type for sample value returned

**Values**

NVML_VALUE_TYPE_DOUBLE = 0
NVML_VALUE_TYPE_UNSIGNED_INT = 1
NVML_VALUE_TYPE_UNSIGNED_LONG = 2
NVML_VALUE_TYPE_UNSIGNED_LONG_LONG = 3
NVML_VALUE_TYPE_SIGNED_LONG_LONG = 4
NVML_VALUE_TYPE_COUNT

```
enum nvmlPerfPolicyType_t
```

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

**Values**

NVML_PERF_POLICY_POWER = 0
   How long did power violations cause the GPU to be below application clocks.
NVML_PERF_POLICY_THERMAL = 1
   How long did thermal violations cause the GPU to be below application clocks.
NVML_PERF_POLICY_SYNC_BOOST = 2
   How long did sync boost cause the GPU to be below application clocks.
NVML_PERF_POLICY_BOARD_LIMIT = 3
   How long did the board limit cause the GPU to be below application clocks.
NVML_PERF_POLICY_LOW_UTILIZATION = 4
   How long did low utilization cause the GPU to be below application clocks.
NVML_PERF_POLICY_RELIABILITY = 5
  How long did the board reliability limit cause the GPU to be below application clocks.

NVML_PERF_POLICY_TOTAL_APP_CLOCKS = 10
  Total time the GPU was held below application clocks by any limiter (0 - 5 above).

NVML_PERF_POLICY_TOTAL_BASE_CLOCKS = 11
  Total time the GPU was held below base clocks.

NVML_PERF_POLICY_COUNT

#define NVML_VALUE_NOT_AVAILABLE (-1)
  Special constant that some fields take when they are not available. Used when only part of the struct is not available.
  Each structure explicitly states when to check for this value.

#define NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE 32
  Buffer size guaranteed to be large enough for pci bus id

#define NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE 16
  Buffer size guaranteed to be large enough for pci bus id for busIdLegacy

#define NVML_DEVICE_PCIE_ID_LEGACY_FMT "%04X:%02X:%02X.0"
  PCI format string for busIdLegacy

#define NVML_DEVICE_PCIE_ID_FMT "%08X:%02X:%02X.0"
  PCI format string for busId

#define NVML_DEVICE_PCIE_ID_FMT_ARGS (pciInfo)->domain, \ (pciInfo)->bus, \ (pciInfo)->device
  Utility macro for filling the pci bus id format from a nvmlPciInfo_t

#define NVML_NVLINK_MAX_LINKS 6
  Maximum number of NvLink links supported
#define NVML_MAX_PHYSICAL_BRIDGE (128)

Maximum limit on Physical Bridges per Board

4.2. Device Enums

enum nvmlEnableState_t

Generic enable/disable enum.

Values

NVML_FEATURE_DISABLED = 0
    Feature disabled.
NVML_FEATURE_ENABLED = 1
    Feature enabled.

enum nvmlBrandType_t

* The Brand of the GPU

Values

NVML_BRAND_UNKNOWN = 0
NVML_BRAND_QUADRO = 1
NVML_BRAND_TESLA = 2
NVML_BRAND_NVS = 3
NVML_BRAND_GRID = 4
NVML_BRAND_GEFORCE = 5
NVML_BRAND_TITAN = 6
NVML_BRAND_COUNT

enum nvmlTemperatureThresholds_t

Temperature thresholds.

Values

NVML_TEMPERATURE_THRESHOLD_SHUTDOWN = 0
NVML_TEMPERATURE_THRESHOLD_SLOWDOWN = 1
NVML_TEMPERATURE_THRESHOLD_MEM_MAX = 2
NVML_TEMPERATURE_THRESHOLD_GPU_MAX = 3
NVML_TEMPERATURE_THRESHOLD_COUNT
enum nvmlTemperatureSensors_t

Temperature sensors.

Values

NVML_TEMPERATURE_GPU = 0
    Temperature sensor for the GPU die.
NVML_TEMPERATURE_COUNT

enum nvmlComputeMode_t

Compute mode.

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

Values

NVML_COMPUTEMODE_DEFAULT = 0
    Default compute mode -- multiple contexts per device.
NVML_COMPUTEMODE_EXCLUSIVE_THREAD = 1
    Support Removed.
NVML_COMPUTEMODE_PROHIBITED = 2
    Compute-prohibited mode -- no contexts per device.
NVML_COMPUTEMODE_EXCLUSIVE_PROCESS = 3
    Compute-exclusive-process mode -- only one context per device, usable from multiple threads at a time.
NVML_COMPUTEMODE_COUNT

enum nvmlMemoryErrorType_t

Memory error types

Values

NVML_MEMORY_ERROR_TYPE_CORRECTED = 0
    A memory error that was corrected For ECC errors, these are single bit errors For Texture memory, these are errors fixed by resend
NVML_MEMORY_ERROR_TYPE_uncorrected = 1
    A memory error that was not corrected For ECC errors, these are double bit errors For Texture memory, these are errors where the resend fails
NVML_MEMORY_ERROR_TYPE_COUNT
    Count of memory error types.
enum nvmlEccCounterType_t

ECC counter types.

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

Values

NVML_VOLATILE_ECC = 0
Volatile counts are reset each time the driver loads.

NVML_AGGREGATE_ECC = 1
Aggregate counts persist across reboots (i.e. for the lifetime of the device).

NVML_ECC_COUNTER_TYPE_COUNT
Count of memory counter types.

enum nvmlClockType_t

Clock types.

All speeds are in Mhz.

Values

NVML_CLOCK_GRAPHICS = 0
Graphics clock domain.

NVML_CLOCK_SM = 1
SM clock domain.

NVML_CLOCK_MEM = 2
Memory clock domain.

NVML_CLOCK_VIDEO = 3
Video encoder/decoder clock domain.

NVML_CLOCK_COUNT
Count of clock types.

enum nvmlClockId_t

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

**NVML_CLOCK_ID_CURRENT** = 0
Current actual clock value.

**NVML_CLOCK_ID_APP_CLOCK_TARGET** = 1
Target application clock.

**NVML_CLOCK_ID_APP_CLOCK_DEFAULT** = 2
Default application clock target.

**NVML_CLOCK_ID_CUSTOMER_BOOST_MAX** = 3
OEM-defined maximum clock rate.

**NVML_CLOCK_ID_COUNT**
Count of Clock Ids.

```c
enum nvmlDriverModel_t
```

Driver models.

Windows only.

Values

**NVML_DRIVER_WDDM** = 0
WDDM driver model -- GPU treated as a display device.

**NVML_DRIVER_WDM** = 1
WDM (TCC) model (recommended) -- GPU treated as a generic device.

```c
enum nvmlPstates_t
```

Allowed PStates.

Values

**NVML_PSTATE_0** = 0
Performance state 0 -- Maximum Performance.

**NVML_PSTATE_1** = 1
Performance state 1.

**NVML_PSTATE_2** = 2
Performance state 2.

**NVML_PSTATE_3** = 3
Performance state 3.

**NVML_PSTATE_4** = 4
Performance state 4.

**NVML_PSTATE_5** = 5
Performance state 5.

**NVML_PSTATE_6** = 6
Performance state 6.
NVML_PSTATE_7 = 7
Performance state 7.

NVML_PSTATE_8 = 8
Performance state 8.

NVML_PSTATE_9 = 9
Performance state 9.

NVML_PSTATE_10 = 10
Performance state 10.

NVML_PSTATE_11 = 11
Performance state 11.

NVML_PSTATE_12 = 12
Performance state 12.

NVML_PSTATE_13 = 13
Performance state 13.

NVML_PSTATE_14 = 14
Performance state 14.

NVML_PSTATE_15 = 15
Performance state 15 -- Minimum Performance.

NVML_PSTATE_UNKNOWN = 32
Unknown performance state.

enum nvmlGpuOperationMode_t

GPU Operation Mode

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

Each GOM is designed to meet specific user needs.

Values

NVML_GOM_ALL_ON = 0
Everything is enabled and running at full speed.

NVML_GOM_COMPUTE = 1
Designed for running only compute tasks. Graphics operations are not allowed.

NVML_GOM_LOW_DP = 2
Designed for running graphics applications that don't require high bandwidth double precision.

enum nvmlInforomObject_t

Available infoROM objects.
Values

NVML_INFOROM_OEM = 0
   An object defined by OEM.
NVML_INFOROM_ECC = 1
   The ECC object determining the level of ECC support.
NVML_INFOROM_POWER = 2
   The power management object.
NVML_INFOROM_COUNT
   This counts the number of infoROM objects the driver knows about.

enum nvmlReturn_t

Return values for NVML API calls.

Values

NVML_SUCCESS = 0
   The operation was successful.
NVML_ERROR_UNINITIALIZED = 1
   NVML was not first initialized with nvmlInit().
NVML_ERROR_INVALID_ARGUMENT = 2
   A supplied argument is invalid.
NVML_ERROR_NOT_SUPPORTED = 3
   The requested operation is not available on target device.
NVML_ERROR_NO_PERMISSION = 4
   The current user does not have permission for operation.
NVML_ERROR_ALREADY_INITIALIZED = 5
   Deprecated: Multiple initializations are now allowed through ref counting.
NVML_ERROR_NOT_FOUND = 6
   A query to find an object was unsuccessful.
NVML_ERROR_INSUFFICIENT_SIZE = 7
   An input argument is not large enough.
NVML_ERROR_INSUFFICIENT_POWER = 8
   A device's external power cables are not properly attached.
NVML_ERROR_DRIVER_NOT_LOADED = 9
   NVIDIA driver is not loaded.
NVML_ERROR_TIMEOUT = 10
   User provided timeout passed.
NVML_ERROR_IRQ_ISSUE = 11
   NVIDIA Kernel detected an interrupt issue with a GPU.
NVML_ERROR_LIBRARY_NOT_FOUND = 12
   NVML Shared Library couldn’t be found or loaded.
NVML_ERROR_FUNCTION_NOT_FOUND = 13
   Local version of NVML doesn’t implement this function.
NVML_ERROR_CORRUPTED_INFOROM = 14
infoROM is corrupted

NVML_ERROR_GPU_IS_LOST = 15
The GPU has fallen off the bus or has otherwise become inaccessible.

NVML_ERROR_RESET_REQUIRED = 16
The GPU requires a reset before it can be used again.

NVML_ERROR_OPERATING_SYSTEM = 17
The GPU control device has been blocked by the operating system/cgroups.

NVML_ERROR_LIB_RM_VERSION_MISMATCH = 18
RM detects a driver/library version mismatch.

NVML_ERROR_IN_USE = 19
An operation cannot be performed because the GPU is currently in use.

NVML_ERROR_MEMORY = 20
Insufficient memory.

NVML_ERROR_NO_DATA = 21
No data.

NVML_ERROR_VGPU_ECC_NOT_SUPPORTED = 22
The requested vgpu operation is not available on target device, because ECC is enabled.

NVML_ERROR_UNKNOWN = 999
An internal driver error occurred.

enum nvmlMemoryLocation_t
See nvmlDeviceGetMemoryErrorCounter

Values

NVML_MEMORY_LOCATION_L1_CACHE = 0
GPU L1 Cache.

NVML_MEMORY_LOCATION_L2_CACHE = 1
GPU L2 Cache.

NVML_MEMORY_LOCATION_DRAM = 2
Turing+ DRAM.

NVML_MEMORY_LOCATION_DEVICE_MEMORY = 2
GPU Device Memory.

NVML_MEMORY_LOCATION_REGISTER_FILE = 3
GPU Register File.

NVML_MEMORY_LOCATION_TEXTURE_MEMORY = 4
GPU Texture Memory.

NVML_MEMORY_LOCATION_TEXTURE_SHM = 5
Shared memory.

NVML_MEMORY_LOCATION_CBU = 6
CBU.
NVML_MEMORY_LOCATION_SRAM = 7
   Turing+ SRAM.
NVML_MEMORY_LOCATION_COUNT
   This counts the number of memory locations the driver knows about.

enum nvmlPageRetirementCause_t

Causes for page retirement

Values

NVML_PAGE RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS = 0
   Page was retired due to multiple single bit ECC error.
NVML_PAGE RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR = 1
   Page was retired due to double bit ECC error.

enum nvmlRestrictedAPI_t

API types that allow changes to default permission restrictions

Values

NVML_RESTRICTED_API_SET_APPLICATION_CLOCKS = 0
   APIs that change application clocks, see nvmlDeviceSetApplicationsClocks and see
   nvmlDeviceResetApplicationsClocks
NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS = 1
   APIs that enable/disable Auto Boosted clocks see
   nvmlDeviceSetAutoBoostedClocksEnabled

#define nvmlFlagDefault 0x00

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

#define nvmlFlagForce 0x01

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

#define nvmlEccBitType_t nvmlMemoryErrorType_t

ECC bit types.
Deprecated See nvmlMemoryErrorType_t for a more flexible type
```
#define NVML_SINGLE_BIT_ECC
NVML_MEMORY_ERROR_TYPE_CORRECTED

Single bit ECC errors
Deprecated Mapped to NVML_MEMORY_ERROR_TYPE_CORRECTED
```

```
#define NVML_DOUBLE_BIT_ECC
NVML_MEMORY_ERROR_TYPE_UNCORRECTED

Double bit ECC errors
Deprecated Mapped to NVML_MEMORY_ERROR_TYPE_UNCORRECTED
```

## 4.3. GRID Enums

```
enum nvmlGpuVirtualizationMode_t

GPU virtualization mode types.
```

### Values

- **NVML_GPU_VIRTUALIZATION_MODE_NONE** = 0
  - Represents Bare Metal GPU.
- **NVML_GPU_VIRTUALIZATION_MODE_PASSTHROUGH** = 1
  - Device is associated with GPU-Passthrough.
- **NVML_GPU_VIRTUALIZATION_MODE_VGPU** = 2
  - Device is associated with vGPU inside virtual machine.
- **NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU** = 3
  - Device is associated with VGX hypervisor in vGPU mode.
- **NVML_GPU_VIRTUALIZATION_MODE_HOST_VSGA** = 4
  - Device is associated with VGX hypervisor in vSGA mode.

## 4.4. Field Value Enums
```c
struct nvmlFieldValue_t

#define NVML_FI_DEV_ECC_CURRENT 1
Current ECC mode. 1=Active. 0=Inactive.
Field Identifiers.
All Identifiers pertain to a device. Each ID is only used once and is guaranteed never to change.

#define NVML_FI_DEV_ECC_PENDING 2
Pending ECC mode. 1=Active. 0=Inactive.

#define NVML_FI_DEV_ECC_SBE_VOL_TOTAL 3
Total single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_TOTAL 4
Total double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_TOTAL 5
Total single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_TOTAL 6
Total double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_L1 7
L1 cache single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_L1 8
L1 cache double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_L2 9
L2 cache single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_L2 10
L2 cache double bit volatile ECC errors.
```
#define NVML_FI_DEV_ECC_SBE_VOL_DEV 11
Device memory single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_DEV 12
Device memory double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_REG 13
Register file single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_REG 14
Register file double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_VOL_TEX 15
Texture memory single bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_TEX 16
Texture memory double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_DBE_VOL_CBU 17
CBU double bit volatile ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_L1 18
L1 cache single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_L1 19
L1 cache double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_L2 20
L2 cache single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_L2 21
L2 cache double bit aggregate (persistent) ECC errors.
#define NVML_FI_DEV_ECC_SBE_AGG_DEV 22
Device memory single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_DEV 23
Device memory double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_REG 24
Register File single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_REG 25
Register File double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_SBE_AGG_TEX 26
Texture memory single bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_TEX 27
Texture memory double bit aggregate (persistent) ECC errors.

#define NVML_FI_DEV_ECC_DBE_AGG_CBU 28
CBU double bit aggregate ECC errors.

#define NVML_FI_DEV_RETIRED_SBE 29
Number of retired pages because of single bit errors.

#define NVML_FI_DEV_RETIRED_DBE 30
Number of retired pages because of double bit errors.

#define NVML_FI_DEV_RETIRED_PENDING 31
If any pages are pending retirement. 1=yes. 0=no.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L0 32
NVLink flow control CRC Error Counter for Lane 0.
#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L1 33
NVLink flow control CRC Error Counter for Lane 1.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L2 34
NVLink flow control CRC Error Counter for Lane 2.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L3 35
NVLink flow control CRC Error Counter for Lane 3.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L4 36
NVLink flow control CRC Error Counter for Lane 4.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L5 37
NVLink flow control CRC Error Counter for Lane 5.

#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_TOTAL 38
NVLink flow control CRC Error Counter total for all Lanes.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L0 39
NVLink data CRC Error Counter for Lane 0.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L1 40
NVLink data CRC Error Counter for Lane 1.
#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L2 41
NVLink data CRC Error Counter for Lane 2.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L3 42
NVLink data CRC Error Counter for Lane 3.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L4 43
NVLink data CRC Error Counter for Lane 4.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L5 44
NVLink data CRC Error Counter for Lane 5.

#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_TOTAL 45
NVLink data CRC Error Counter total for all Lanes.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L0 46
NVLink Replay Error Counter for Lane 0.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L1 47
NVLink Replay Error Counter for Lane 1.

#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L2 48
NVLink Replay Error Counter for Lane 2.
#define NVML_FI_DEV_NVMLINK_REPLAY_ERROR_COUNT_L3 49
NVLink Replay Error Counter for Lane 3.

#define NVML_FI_DEV_NVMLINK_REPLAY_ERROR_COUNT_L4 50
NVLink Replay Error Counter for Lane 4.

#define NVML_FI_DEV_NVMLINK_REPLAY_ERROR_COUNT_L5 51
NVLink Replay Error Counter for Lane 5.

#define NVML_FI_DEV_NVMLINK_REPLAY_ERROR_COUNT_TOTAL 52
NVLink Replay Error Counter total for all Lanes.

#define NVML_FI_DEV_NVMLINK_RECOVERY_ERROR_COUNT_L0 53
NVLink Recovery Error Counter for Lane 0.

#define NVML_FI_DEV_NVMLINK_RECOVERY_ERROR_COUNT_L1 54
NVLink Recovery Error Counter for Lane 1.

#define NVML_FI_DEV_NVMLINK_RECOVERY_ERROR_COUNT_L2 55
NVLink Recovery Error Counter for Lane 2.

#define NVML_FI_DEV_NVMLINK_RECOVERY_ERROR_COUNT_L3 56
NVLink Recovery Error Counter for Lane 3.
#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L4 57
NVLink Recovery Error Counter for Lane 4.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L5 58
NVLink Recovery Error Counter for Lane 5.

#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_TOTAL 59
NVLink Recovery Error Counter total for all Lanes.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L0 60
NVLink Bandwidth Counter for Counter Set 0, Lane 0.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L1 61
NVLink Bandwidth Counter for Counter Set 0, Lane 1.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L2 62
NVLink Bandwidth Counter for Counter Set 0, Lane 2.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L3 63
NVLink Bandwidth Counter for Counter Set 0, Lane 3.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L4 64
NVLink Bandwidth Counter for Counter Set 0, Lane 4.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L5 65
NVLink Bandwidth Counter for Counter Set 0, Lane 5.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_TOTAL 66
NVLink Bandwidth Counter Total for Counter Set 0, All Lanes.
#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L0 67
NVLink Bandwidth Counter for Counter Set 1, Lane 0.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L1 68
NVLink Bandwidth Counter for Counter Set 1, Lane 1.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L2 69
NVLink Bandwidth Counter for Counter Set 1, Lane 2.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L3 70
NVLink Bandwidth Counter for Counter Set 1, Lane 3.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L4 71
NVLink Bandwidth Counter for Counter Set 1, Lane 4.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L5 72
NVLink Bandwidth Counter for Counter Set 1, Lane 5.

#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_TOTAL 73
NVLink Bandwidth Counter Total for Counter Set 1, All Lanes.

#define NVML_FI_DEV_PERF_POLICY_POWER 74
Perf Policy Counter for Power Policy.

#define NVML_FI_DEV_PERF_POLICY_THERMAL 75
Perf Policy Counter for Thermal Policy.

#define NVML_FI_DEV_PERF_POLICY_SYNC_BOOST 76
Perf Policy Counter for Sync boost Policy.

#define NVML_FI_DEV_PERF_POLICY_BOARD_LIMIT 77
Perf Policy Counter for Board Limit.
#define NVML_FI_DEV_PERF_POLICY_LOW_UTILIZATION 78
Perf Policy Counter for Low GPU Utilization Policy.

#define NVML_FI_DEV_PERF_POLICY_RELIABILITY 79
Perf Policy Counter for Reliability Policy.

#define NVML_FI_DEV_PERF_POLICY_TOTAL_APPCLOCKS 80
Perf Policy Counter for Total App Clock Policy.

#define NVML_FI_DEV_PERF_POLICY_TOTAL_BASECLOCKS 81
Perf Policy Counter for Total Base Clocks Policy.

#define NVML_FI_DEV_MEMORY_TEMP 82
Memory temperature for the device.

#define NVML_FI_DEV_TOTALENERGYCONSUMPTION 83
Total energy consumption for the GPU in mJ since the driver was last reloaded.

#define NVML_FI_DEV_NVLINKSPEED_MBPS_L0 84
NVLink Speed in MBps for Link 0.

#define NVML_FI_DEV_NVLINKSPEED_MBPS_L1 85
NVLink Speed in MBps for Link 1.

#define NVML_FI_DEV_NVLINKSPEED_MBPS_L2 86
NVLink Speed in MBps for Link 2.

#define NVML_FI_DEV_NVLINKSPEED_MBPS_L3 87
NVLink Speed in MBps for Link 3.
#define NVML_FI_DEV_NVLINK_SPEED_Mbps_L4 88
NVLink Speed in MBps for Link 4.

#define NVML_FI_DEV_NVLINK_SPEED_Mbps_L5 89
NVLink Speed in MBps for Link 5.

#define NVML_FI_DEV_NVLINK_SPEED_Mbps_COMMON 90
Common NVLink Speed in MBps for active links.

#define NVML_FI_DEV_NVLINK_LINK_COUNT 91
Number of NVLinks present on the device.

#define NVML_FI_DEV_RETIRED_PENDING_SBE 92
If any pages are pending retirement due to SBE. 1=yes. 0=no.

#define NVML_FI_DEV_RETIRED_PENDING_DBE 93
If any pages are pending retirement due to DBE. 1=yes. 0=no.

#define NVML_FI_DEV_PCIE_REPLAY_COUNTER 94
PCIe replay counter.

#define NVML_FI_DEV_PCIE_REPLAY_ROLLOVER_COUNTER 95
PCIe replay rollover counter.

#define NVML_FI_MAX 96
One greater than the largest field ID defined above.

4.5. Unit Structs
struct nvmlHwbcEntry_t
struct nvmlLedState_t
struct nvmlUnitInfo_t
struct nvmlPSUInfo_t
struct nvmlUnitFanInfo_t
struct nvmlUnitFanSpeeds_t
enum nvmlFanState_t

Fan state enum.

Values

NVML_FAN_NORMAL = 0
    Fan is working properly.
NVML_FAN_FAILED = 1
    Fan has failed.

enum nvmlLedColor_t

Led color enum.

Values

NVML_LED_COLOR_GREEN = 0
    GREEN, indicates good health.
NVML_LED_COLOR_AMBER = 1
    AMBER, indicates problem.

4.6. Accounting Statistics

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

> All accounting statistics and accounting mode live in nvidia driver and reset to default (Disabled) when driver unloads. It is advised to run with persistence mode enabled.
Enabling accounting mode has no negative impact on the GPU performance.

```
struct nvmlAccountingStats_t

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

**Parameters**

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

**Returns**

- NVML_SUCCESS if the mode has been successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode are NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

Queries the state of per process accounting mode.

KEPLER_OR_NEWER%

See `nvmlDeviceGetAccountingStats` for more details. See `nvmlDeviceSetAccountingMode`

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

**Parameters**

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

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

Description

Queries process’s accounting stats.

KEPLER_OR_NEWER%

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

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

- Accounting Mode needs to be on. See `nvmlDeviceGetAccountingMode`.
- Only compute and graphics applications stats can be queried. Monitoring applications stats can’t be queried since they don’t contribute to GPU utilization.
- In case of pid collision stats of only the latest process (that terminated last) will be reported

See also:

`nvmlDeviceGetAccountingBufferSize`

```c
nvmlReturn_t nvmlDeviceGetAccountingPids
(nvmlDevice_t device, unsigned int *count, unsigned int *pids)
```

Parameters

- `device`
  The identifier of the target device
count
Reference in which to provide the pids array size, and to return the number of elements ready to be queried

pids
Reference in which to return list of process ids

Returns

- NVML_SUCCESS if pids were successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or count is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature or accounting mode is disabled
- NVML_ERROR_INSUFFICIENT_SIZE if count is too small (count is set to expected value)
- NVML_ERROR_UNKNOWN on any unexpected error

Description

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

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

For more details see nvmlDeviceGetAccountingStats.

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

See also:

nvmlDeviceGetAccountingBufferSize

nvmlReturn_t nvmlDeviceGetAccountingBufferSize
(nvmlDevice_t device, unsigned int *bufferSize)

Parameters

device
The identifier of the target device
bufferSize
Reference in which to provide the size (in number of elements) of the circular buffer for accounting stats.

Returns
- NVML_SUCCESS if buffer size was successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or bufferSize is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature or accounting mode is disabled
- NVML_ERROR_UNKNOWN on any unexpected error

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

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

See also:
nvmlDeviceGetAccountingStats
nvmlDeviceGetAccountingPids

nvmlReturn_t nvmlDeviceSetAccountingMode
(nvmlDevice_t device, nvmlEnableState_t mode)

Parameters
- device
  The identifier of the target device
- mode
  The target accounting mode

Returns
- NVML_SUCCESS if the new mode has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device or mode are invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

Enables or disables per process accounting.

KEPLER_OR_NEWER% Requires root/admin permissions.

- This setting is not persistent and will default to disabled after driver unloads. Enable persistence mode to be sure the setting doesn't switch off to disabled.
- Enabling accounting mode has no negative impact on the GPU performance.
- Disabling accounting clears all accounting pids information.

See `nvmlDeviceGetAccountingMode` See `nvmlDeviceGetAccountingStats` See `nvmlDeviceClearAccountingPids`

```c
nvmlReturn_t nvmlDeviceClearAccountingPids(nvmlDevice_t device)
```

**Parameters**

device
   The identifier of the target device

**Returns**

- NVML_SUCCESS if accounting information has been cleared
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device are invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

Clears accounting information about all processes that have already terminated.

KEPLER_OR_NEWER% Requires root/admin permissions.

See `nvmlDeviceGetAccountingMode` See `nvmlDeviceGetAccountingStats` See `nvmlDeviceSetAccountingMode`
4.7. Vgpu Constants

#define NVML_GRID_LICENSE_BUFFER_SIZE 128
Buffer size guaranteed to be large enough for nvmlVgpuTypeGetLicense

#define NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION 0:0
Macros for pGPU’s virtualization capabilities bitfield.

4.8. Vgpu Enum

enum nvmlVgpuVmIdType_t
Types of VM identifiers

Values

NVML_VGPU_VM_ID_DOMAIN_ID = 0
VM ID represents DOMAIN ID.
NVML_VGPU_VM_ID_UUID = 1
VM ID represents UUID.

enum nvmlVgpuGuestInfoState_t
vGPU GUEST info state.

Values

NVML_VGPU_INSTANCE_GUEST_INFO_STATE_UNINITIALIZED = 0
Guest-dependent fields uninitialized.
NVML_VGPU_INSTANCE_GUEST_INFO_STATE_INITIALIZED = 1
Guest-dependent fields initialized.

enum nvmlGridLicenseFeatureCode_t
GRID license feature code
**4.9. Vgpu Structs**

- struct nvmlVgpuInstanceUtilizationSample_t
- struct nvmlVgpuProcessUtilizationSample_t
- struct nvmlProcessUtilizationSample_t
- struct nvmlGridLicensableFeature_t
- struct nvmlGridLicensableFeatures_t

**4.10. Encoder Structs**

- struct nvmlEncoderSessionInfo_t
- enum nvmlEncoderType_t

  Represents type of encoder for capacity can be queried

**Values**

- NVML_ENCODER_QUERY_H264 = 0
  - H264 encoder.
- NVML_ENCODER_QUERY_HEVC = 1
  - HEVC encoder.

**4.11. Frame Buffer Capture Structures**
struct nvmlFBCStats_t

struct nvmlFBCSessionInfo_t

define nvmlFBCSessionType_t

Represents frame buffer capture session type

Values

NVML_FBC_SESSION_TYPE_UNKNOWN = 0

Unknown.

NVML_FBC_SESSION_TYPE_TOSYS

ToSys.

NVML_FBC_SESSION_TYPE_CUDA

Cuda.

NVML_FBC_SESSION_TYPE_VID

Vid.

NVML_FBC_SESSION_TYPE_HWENC

HEnc.

#define NVML_NVFBC_SESSION_FLAG_DIFFMAP_ENABLED

0x00000001

Bit specifying differential map state.

#define NVML_NVFBC_SESSION_FLAG_CLASSIFICATIONMAP_ENABLED

0x00000002

Bit specifying classification map state.

#define NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_NO_WAIT

0x00000004

Bit specifying if capture was requested as non-blocking call.
#define NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_INFINITE 0x00000008
Bit specifying if capture was requested as blocking call.

#define NVML_NVFBC_SESSION_FLAG_CAPTURE_WITH_WAIT_TIMEOUT 0x00000010
Bit specifying if capture was requested as blocking call with timeout period.

4.12. definitions related to the drain state

defined enum nvmlDetachGpuState_t
Is the GPU device to be removed from the kernel by nvmlDeviceRemoveGpu()

Values
NVML_DETACH_GPU_KEEP = 0
NVML_DETACH_GPU_REMOVE

defined enum nvmlPcieLinkState_t
Parent bridge PCIe link state requested by nvmlDeviceRemoveGpu()

Values
NVML_PCIE_LINK_KEEP = 0
NVML_PCIE_LINK_SHUT_DOWN

4.13. Initialization and Cleanup

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

Returns

‣ NVML_SUCCESS if NVML has been properly initialized
‣ NVML_ERROR_DRIVER_NOT_LOADED if NVIDIA driver is not running
‣ NVML_ERROR_NO_PERMISSION if NVML does not have permission to talk to the driver
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description

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

nvmlInit_v3 introduces a "flags" argument, that allows passing boolean values modifying the behaviour of nvmlInit().

In NVML 5.319 new nvmlInit_v2 has replaced nvmlInit"_v1" (default in NVML 4.304 and older) that did initialize all GPU devices in the system.

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

To contrast nvmlInit_v2 with nvmlInit"_v1", NVML 4.304 nvmlInit"_v1" will fail when any detected GPU is in a bad or unstable state.

ALL_PRODUCTS%

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

nvmlReturn_t nvmlInitWithFlags (unsigned int flags)

Parameters

flags
  behaviour modifier flags

Returns

‣ NVML_SUCCESS if NVML has been properly initialized
‣ NVML_ERROR_DRIVER_NOT_LOADED if NVIDIA driver is not running
- NVML_ERROR_NO_PERMISSION if NVML does not have permission to talk to the driver
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**
nvmlInitWithFlags is a variant of nvmlInit(), that allows passing a set of boolean values modifying the behaviour of nvmlInit(). Other than the "flags" parameter it is completely similar to nvmlInit.

**nvmlReturn_t nvmlShutdown (void)**

**Returns**
- NVML_SUCCESS if NVML has been properly shut down
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**
Shut down NVML by releasing all GPU resources previously allocated with nvmlInit(). This method should be called after NVML work is done, once for each call to nvmlInit(). A reference count of the number of initializations is maintained. Shutdown only occurs when the reference count reaches zero. For backwards compatibility, no error is reported if nvmlShutdown() is called more times than nvmlInit().

#define NVML_INIT_FLAG_NO_GPUS 1
Don’t fail nvmlInit() when no GPUs are found.

#define NVML_INIT_FLAG_NO_ATTACH 2
Don’t attach GPUs.

**4.14. Error reporting**
This chapter describes helper functions for error reporting routines.
const DECLDIR char *nvmlErrorString (nvmlReturn_t result)

Parameters
result
    NVML error code to convert

Returns
String representation of the error.

Description
Helper method for converting NVML error codes into readable strings.

4.15. Constants

#define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16
Buffer size guaranteed to be large enough for nvmlDeviceGetInforomVersion and nvmlDeviceGetInforomImageVersion

#define NVML_DEVICE_UUID_BUFFER_SIZE 80
Buffer size guaranteed to be large enough for nvmlDeviceGetUUID

#define NVML_DEVICE_PART_NUMBER_BUFFER_SIZE 80
Buffer size guaranteed to be large enough for nvmlDeviceGetBoardPartNumber

#define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80
Buffer size guaranteed to be large enough for nvmlSystemGetDriverVersion

#define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80
Buffer size guaranteed to be large enough for nvmlSystemGetNVMLVersion
#define NVML_DEVICE_NAME_BUFFER_SIZE 64
Buffer size guaranteed to be large enough for `nvmlDeviceGetName`

#define NVML_DEVICE_SERIAL_BUFFER_SIZE 30
Buffer size guaranteed to be large enough for `nvmlDeviceGetSerial`

#define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32
Buffer size guaranteed to be large enough for `nvmlDeviceGetVbiosVersion`

**4.16. System Queries**

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

`nvmlReturn_t nvmlSystemGetDriverVersion (char *version, unsigned int length)`

**Parameters**

- *version*
  - Reference in which to return the version identifier

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

**Returns**

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small

**Description**

Retrieves the version of the system’s graphics driver.

The version identifier is an alphanumeric string. It will not exceed 80 characters in length (including the NULL terminator). See `nvmlConstants::NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE`.
nvmlReturn_t nvmlSystemGetNVMLVersion (char *version, unsigned int length)

Parameters

version
  Reference in which to return the version identifier
length
  The maximum allowed length of the string returned in version

Returns

‣ NVML_SUCCESS if version has been set
‣ NVML_ERROR_INVALID_ARGUMENT if version is NULL
‣ NVML_ERROR_INSUFFICIENT_SIZE if length is too small

Description
Retrieves the version of the NVML library.

The version identifier is an alphanumeric string. It will not exceed 80 characters in length (including the NULL terminator). See
nvmlConstants::NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE.

nvmlReturn_t nvmlSystemGetCudaDriverVersion (int *cudaDriverVersion)

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

Description
Retrieves the version of the CUDA driver.

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.

```
nvmlReturn_t nvmlSystemGetCudaDriverVersion_v2 (int *cudaDriverVersion)
```

### 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

### Description

Retrieves the version of the CUDA driver from the shared library.

**ALL_PRODUCTS%**

The returned CUDA driver version by calling cuDriverGetVersion()

```
nvmlReturn_t nvmlSystemGetProcessName (unsigned int pid, char *name, unsigned int length)
```

### Parameters

- **pid**
  - The identifier of the process
- **name**
  - Reference in which to return the process name
- **length**
  - The maximum allowed length of the string returned in name

### Returns

- NVML_SUCCESS if name has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if name is NULL or length is 0.
- NVML_ERROR_NOT_FOUND if process doesn't exist.
- NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation.
- NVML_ERROR_UNKNOWN on any unexpected error.

**Description**

Gets name of the process with provided process id.

**ALL_PRODUCTS**

Returned process name is cropped to provided length. name string is encoded in ANSI.

```c
#define NVML_CUDA_DRIVER_VERSION_MAJOR 
((v)/1000)
```

Macros for converting the CUDA driver version number to Major and Minor version numbers.

### 4.17. Unit Queries

This chapter describes 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()`.

```c
nvmlReturn_t nvmlUnitGetCount (unsigned int *unitCount)
```

**Parameters**

- `unitCount`
  - Reference in which to return the number of units

**Returns**

- NVML_SUCCESS if unitCount has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if unitCount is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

Retrieves the number of units in the system.
nvmlReturn_t nvmlUnitGetHandleByIndex (unsigned int index, nvmlUnit_t *unit)

Parameters

index
The index of the target unit, >= 0 and < unitCount

unit
Reference in which to return the unit handle

Returns

- NVML_SUCCESS if unit has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if index is invalid or unit is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Acquire the handle for a particular unit, based on its index.

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.

nvmlReturn_t nvmlUnitGetUnitInfo (nvmlUnit_t unit, nvmlUnitInfo_t *info)

Parameters

unit
The identifier of the target unit

info
Reference in which to return the unit information

Returns

- NVML_SUCCESS if info has been populated
• NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
• NVML_ERROR_INVALID_ARGUMENT if unit is invalid or info is NULL

Description
Retrieves the static information associated with a unit.

S_CLASS%
See nvmlUnitInfo_t for details on available unit info.

nvmlReturn_t nvmlUnitGetLedState (nvmlUnit_t unit,
nvmlLedState_t *state)

Parameters
unit
The identifier of the target unit
state
Reference in which to return the current LED state

Returns
• NVML_SUCCESS if state has been set
• NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
• NVML_ERROR_INVALID_ARGUMENT if unit is invalid or state is NULL
• NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
• NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the LED state associated with this unit.

S_CLASS%
See nvmlLedState_t for details on allowed states.

See also:
nvmlUnitSetLedState()
nvmlReturn_t nvmlUnitGetPsuInfo (nvmlUnit_t unit, nvmlPSUInfo_t *psu)

Parameters

unit
  The identifier of the target unit

psu
  Reference in which to return the PSU information

Returns

‣ NVML_SUCCESS if psu has been populated
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if unit is invalid or psu is NULL
‣ NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the PSU stats for the unit.

S_CLASS%

See nvmlPSUInfo_t for details on available PSU info.

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

Parameters

unit
  The identifier of the target unit

type
  The type of reading to take

temp
  Reference in which to return the intake temperature

Returns

‣ NVML_SUCCESS if temp has been populated
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if unit or type is invalid or temp is NULL
‣ NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
NVML_ERROR_UNKNOWN on any unexpected error

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

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

```c
nvmlReturn_t nvmlUnitGetFanSpeedInfo (nvmlUnit_t unit, nvmlUnitFanSpeeds_t *fanSpeeds)
```

**Parameters**
- **unit**
  The identifier of the target unit
- **fanSpeeds**
  Reference in which to return the fan speed information

**Returns**
- NVML_SUCCESS if fanSpeeds has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if unit is invalid or fanSpeeds is NULL
- NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**
Retrieves the fan speed readings for the unit.

S_CLASS%
See `nvmlUnitFanSpeeds_t` for details on available fan speed info.

```c
nvmlReturn_t nvmlUnitGetDevices (nvmlUnit_t unit, unsigned int *deviceCount, nvmlDevice_t *devices)
```

**Parameters**
- **unit**
  The identifier of the target unit
**deviceCount**
Reference in which to provide the devices array size, and to return the number of attached GPU devices

**devices**
Reference in which to return the references to the attached GPU devices

**Returns**
- NVML_SUCCESS if deviceCount and devices have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if deviceCount indicates that the devices array is too small
- NVML_ERROR_INVALID_ARGUMENT if unit is invalid, either of deviceCount or devices is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**
Retrieves the set of GPU devices that are attached to the specified unit.

S_CLASS%
The deviceCount argument is expected to be set to the size of the input devices array.

gmlReturn_t nvmlSystemGetHicVersion (unsigned int *hwbcCount, nvmlHwbcEntry_t *hwbcEntries)

**Parameters**

**hwbcCount**
Size of hwbcEntries array

**hwbcEntries**
Array holding information about hwbc

**Returns**
- NVML_SUCCESS if hwbcCount and hwbcEntries have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if either hwbcCount or hwbcEntries is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if hwbcCount indicates that the hwbcEntries array is too small
Description
Retrieves the IDs and firmware versions for any Host Interface Cards (HICs) in the system.

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

4.18. Device Queries
This chapter describes that queries that NVML can perform against each device. In each case the device is identified with an nvmlDevice_t handle. This handle is obtained by calling one of nvmlDeviceGetHandleByIndex(), nvmlDeviceGetHandleBySerial(), nvmlDeviceGetHandleByPciBusId(). or nvmlDeviceGetHandleByUUID().

nvmlReturn_t nvmlDeviceGetCount (unsigned int *deviceCount)

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

Returns
- NVML_SUCCESS if deviceCount has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if deviceCount is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

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

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

**nvmlReturn_t nvmlDeviceGetHandleByIndex (unsigned int index, nvmlDevice_t *device)**

**Parameters**

- **index**
  
  The index of the target GPU, >= 0 and < accessibleDevices

- **device**
  
  Reference in which to return the device handle

**Returns**

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

**Description**

Acquire the handle for a particular device, based on its index.

**ALL_PRODUCTS%**

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

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

Note: The NVML index may not correlate with other APIs, such as the CUDA device index.
Starting from NVML 5, this API causes NVML to initialize the target GPU NVML may initialize additional GPUs if:

- The target GPU is an SLI slave

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

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

See also:

- `nvmlDeviceGetIndex`
- `nvmlDeviceGetCount`

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

#### Parameters

- **serial**
  - The board serial number of the target GPU

- **device**
  - Reference in which to return the device handle

#### Returns

- `NVML_SUCCESS` if device has been set
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if serial is invalid, device is NULL or more than one device has the same serial (dual GPU boards)
- `NVML_ERROR_NOT_FOUND` if serial does not match a valid device on the system
- `NVML_ERROR_INSUFFICIENT_POWER` if any attached devices have improperly attached external power cables
- `NVML_ERROR_IRQ_ISSUE` if NVIDIA kernel detected an interrupt issue with the attached GPUs
- `NVML_ERROR_GPU_IS_LOST` if any GPU has fallen off the bus or is otherwise inaccessible
Description
Acquire the handle for a particular device, based on its board serial number.

FERMI_OR_NEWER%

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

Deprecated Since more than one GPU can exist on a single board this function is deprecated in favor of `nvmlDeviceGetHandleByUUID`. For dual GPU boards this function will return NVML_ERROR_INVALID_ARGUMENT.

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

See also:

`nvmlDeviceGetSerial`

`nvmlDeviceGetHandleByUUID`

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

Parameters

`uuid`
The UUID of the target GPU

`device`
Reference in which to return the device handle

Returns

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

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

**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

```c
nvmlReturn_t nvmlDeviceGetHandleByPciBusId (const char *pciBusId, nvmlDevice_t *device)
```

**Parameters**

- **pciBusId**
  - The PCI bus id of the target GPU
- **device**
  - Reference in which to return the device handle

**Returns**

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

**Description**

Acquire the handle for a particular device, based on its PCI bus id.

**ALL_PRODUCTS**
This value corresponds to the `nvmlPciInfo_t::busId` returned by `nvmlDeviceGetPciInfo()`.

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

- The target GPU is an SLI slave

---

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

---

```c
nvmlReturn_t nvmlDeviceGetName (nvmlDevice_t device, char *name, unsigned int length)
```

**Parameters**

- `device`:
  The identifier of the target device
- `name`:
  Reference in which to return the product name
- `length`:
  The maximum allowed length of the string returned in `name`

**Returns**

- `NVML_SUCCESS` if `name` has been set
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if device is invalid, or `name` is `NULL`
- `NVML_ERROR_INSUFFICIENT_SIZE` if length is too small
- `NVML_ERROR_GPU_IS_LOST` if the target GPU has fallen off the bus or is otherwise inaccessible
- `NVML_ERROR_UNKNOWN` on any unexpected error

**Description**

Retrieves the name of this device.

- `ALL_PRODUCTS%`
  The name is an alphanumeric string that denotes a particular product, e.g. Tesla C2070. It will not exceed 64 characters in length (including the NULL terminator). See `nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE`. 
nvmlReturn_t nvmlDeviceGetBrand (nvmlDevice_t device, nvmlBrandType_t *type)

Parameters

device
The identifier of the target device

type
Reference in which to return the product brand type

Returns

- NVML_SUCCESS if name has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or type is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the brand of this device.

ALL_PRODUCTS%

The type is a member of nvmlBrandType_t defined above.

nvmlReturn_t nvmlDeviceGetIndex (nvmlDevice_t device, unsigned int *index)

Parameters

device
The identifier of the target device

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

Returns

- NVML_SUCCESS if index has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or index is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
• NVML_ERROR_UNKNOWN on any unexpected error

**Description**

Retrieves the NVML index of this device.

**ALL_PRODUCTS**

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

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

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

**See also:**

`nvmlDeviceGetHandleByIndex()`

`nvmlDeviceGetCount()`

```c
nvmlReturn_t nvmlDeviceGetSerial (nvmlDevice_t device, char *serial, unsigned int length)
```

**Parameters**

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

- **serial**
  - Reference in which to return the board/module serial number

- **length**
  - The maximum allowed length of the string returned in serial

**Returns**

- NVML_SUCCESS if serial has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or serial is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
**Description**
Retrieves the globally unique board serial number associated with this device's board.

INFOROM_SUPPORT%

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.

```
vmlReturn_t nvmlDeviceGetCpuAffinity (nvmlDevice_t device, unsigned int cpuSetSize, unsignedlong *cpuSet)
```

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

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

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

KEPLER_OR_NEWER% Supported on Linux only.
nvmlReturn_t nvmlDeviceSetCpuAffinity (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

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

Description

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

KEPLER_OR_NEWER% Supported on Linux only.

nvmlReturn_t nvmlDeviceClearCpuAffinity (nvmlDevice_t device)

Parameters

device
The identifier of the target device

Returns

- NVML_SUCCESS if the calling process has been successfully unbound
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_UNKNOWN on any unexpected error
Description
Clear all affinity bindings for 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.
KEPLER_OR_NEWER% Supported on Linux only.

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

Parameters
- **device1**
  The identifier of the first device
- **device2**
  The identifier of the second device
- **pathInfo**
  A `nvmlGpuTopologyLevel_t` that gives the path type

Returns
- NVML_SUCCESS if pathInfo has been set
- NVML_ERROR_INVALID_ARGUMENT if device1, or device2 is invalid, or pathInfo is NULL
- NVML_ERROR_NOT_SUPPORTED if the device or OS does not support this feature
- NVML_ERROR_UNKNOWN an error has occurred in underlying topology discovery

Description
Retrieve the common ancestor for two devices ALL_PRODUCTS% Supported on Linux only.

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

Parameters
- **device**
  The identifier of the first device
level
The `nvmlGpuTopologyLevel_t` level to search for other GPUs

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

deviceArray
An array of device handles for GPUs found at level

Returns

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

Description
Retrieve the set of GPUs that are nearest to a given device at a specific interconnectivity level ALL_PRODUCTS% Supported on Linux only.

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

Parameters

cpuNumber
The CPU number

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

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

Returns

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

**Description**
Retrieve the set of GPUs that have a CPU affinity with the given CPU number
ALL_PRODUCTS% Supported on Linux only.

```c
nvmlReturn_t nvmlDeviceGetP2PStatus
(nvmlDevice_t device1, nvmlDevice_t device2,
nvmlGpuP2PCapsIndex_t p2pIndex, nvmlGpuP2PStatus_t *p2pStatus)
```

**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

**Description**
Retrieve the status for a given p2p capability index between a given pair of GPU
nvmlReturn_t nvmlDeviceGetUUID (nvmlDevice_t device, char *uuid, unsigned int length)

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

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

The UUID is a globally unique identifier. It is the only available identifier for pre-Fermi-architecture products. It does NOT correspond to any identifier printed on the board. It will not exceed 80 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE.

nvmlReturn_t nvmlDeviceGetMinorNumber (nvmlDevice_t device, unsigned int *minorNumber)

Parameters
device
The identifier of the target device
**minorNumber**
Reference in which to return the minor number for the device

**Returns**
- NVML_SUCCESS if the minor number is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or minorNumber is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

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

**ALL PRODUCTS**% Supported only for Linux

```c
nvmlReturn_t nvmlDeviceGetBoardPartNumber(nvmlDevice_t device, char *partNumber, unsigned int length)
```

**Parameters**
- **device**
  Identifier of the target device
- **partNumber**
  Reference to the buffer to return
- **length**
  Length of the buffer reference

**Returns**
- NVML_SUCCESS if partNumber has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_SUPPORTED if the needed VBIOS fields have not been filled
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or serial is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error
Description
Retrieves the the device board part number which is programmed into the board’s InfoROM

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

Parameters

device
The identifier of the target device

object
The target infoROM object

version
Reference in which to return the infoROM version

length
The maximum allowed length of the string returned in version

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_NOT_SUPPORTED if the device does not have an infoROM
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the version information for the device’s infoROM object.

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

See nvmlInforomObject_t for details on the available infoROM objects.
See also:

nvmlDeviceGetInforomImageVersion

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

Parameters

device
  The identifier of the target device

version
  Reference in which to return the infoROM image version

length
  The maximum allowed length of the string returned in version

Returns

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_NOT_SUPPORTED if the device does not have an infoROM
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the global infoROM image version

INFOROM_SUPPORT%

Image version just like VBIOS version uniquely describes the exact version of the infoROM flashed on the board in contrast to infoROM object version which is only an indicator of supported features. Version string will not exceed 16 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE.

See also:

nvmlDeviceGetInforomVersion
nvmlReturn_t
nvmlDeviceGetInfoRomConfigurationChecksum(nvmlDevice_t device, unsigned int *checksum)

Parameters

device
The identifier of the target device

checksum
Reference in which to return the infoROM configuration checksum

Returns

- NVML_SUCCESS if checksum has been set
- NVML_ERROR_CORRUPTED_INFOROM if the device's checksum couldn't be retrieved due to infoROM corruption
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if checksum is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

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

INFOROM_SUPPORT%

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

nvmlReturn_t nvmlDeviceValidateInfoRom(nvmlDevice_t device)

Parameters

device
The identifier of the target device
Returns

- NVML_SUCCESS if infoROM is not corrupted
- NVML_ERROR_CORRUPTED_INFOROM if the device’s infoROM is corrupted
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Reads the infoROM from the flash and verifies the checksums.

```
INFOROM_SUPPORT%
```

```
nvmlReturn_t nvmlDeviceGetDisplayMode (nvmlDevice_t device, nvmlEnableState_t *display)
```

Parameters

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

Returns

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

Description

Retrieves the display mode for the device.

```
ALL_PRODUCTS%
```

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

See `nvmlEnableState_t` for details on allowed modes.
nvmlReturn_t nvmlDeviceGetDisplayActive
(nvmlDevice_t device, nvmlEnableState_t *isActive)

Parameters

device
   The identifier of the target device

isActive
   Reference in which to return the display active state

Returns

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

Description
Retrieves the display active state for the device.

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

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

See nvmlEnableState_t for details on allowed modes.

nvmlReturn_t nvmlDeviceGetPersistenceMode
(nvmlDevice_t device, nvmlEnableState_t *mode)

Parameters

device
   The identifier of the target device

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

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

Description

Retrieves the persistence mode associated with this device.

ALL PRODUCTS For Linux only.

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

See `nvmlEnableState_t` for details on allowed modes.

See also:

nvmlDeviceSetPersistenceMode()

```
nvmlReturn_t nvmlDeviceGetPciInfo (nvmlDevice_t device, nvmlPciInfo_t *pci)
```

Parameters

- **device**
  - The identifier of the target device
- **pci**
  - Reference in which to return the PCI info

Returns

- NVML_SUCCESS if pci has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pci is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error
Description
Retrieves the PCI attributes of this device.

See `nvmlPcieInfo_t` for details on the available PCI info.

`nvmlReturn_t nvmlDeviceGetMaxPcieLinkGeneration(nvmlDevice_t device, unsigned int *maxLinkGen)`

Parameters
- `device` The identifier of the target device
- `maxLinkGen` Reference in which to return the max PCIe link generation

Returns
- NVML_SUCCESS if maxLinkGen has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or maxLinkGen is null
- NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the maximum PCIe link generation possible with this device and system
I.E. for a generation 2 PCIe device attached to a generation 1 PCIe bus the max link generation this function will report is generation 1.

`nvmlReturn_t nvmlDeviceGetMaxPcieLinkWidth(nvmlDevice_t device, unsigned int *maxLinkWidth)`

Parameters
- `device` The identifier of the target device
maxLinkWidth
Reference in which to return the max PCIe link generation

Returns
- NVML_SUCCESS if maxLinkWidth has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or maxLinkWidth is null
- NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the maximum PCIe link width possible with this device and system
I.E. for a device with a 16x PCIe bus width attached to a 8x PCIe system bus this function will report a max link width of 8.

FERMI_OR_NEWER%

nvmlReturn_t nvmlDeviceGetCurrPcieLinkGeneration
(nvmlDevice_t device, unsigned int *currLinkGen)

Parameters
device
The identifier of the target device
currLinkGen
Reference in which to return the current PCIe link generation

Returns
- NVML_SUCCESS if currLinkGen has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or currLinkGen is null
- NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error
Description
Retrieves the current PCIe link generation
FERMI_OR_NEWER%

nvmlReturn_t nvmlDeviceGetCurrPcieLinkWidth
(nvmlDevice_t device, unsigned int *currLinkWidth)

Parameters
device
The identifier of the target device
currLinkWidth
Reference in which to return the current PCIe link generation

Returns
‣ NVML_SUCCESS if currLinkWidth has been populated
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if device is invalid or currLinkWidth is null
‣ NVML_ERROR_NOT_SUPPORTED if PCIe link information is not available
‣ NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the current PCIe link width
FERMI_OR_NEWER%

nvmlReturn_t nvmlDeviceGetPcieThroughput
(nvmlDevice_t device, nvmlPcieUtilCounter_t counter, unsigned int *value)

Parameters
device
The identifier of the target device
counter
The specific counter that should be queried nvmlPcieUtilCounter_t
value
Reference in which to return throughput in KB/s

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

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

MAXWELL_OR_NEWER%
This method is not supported in virtual machines running virtual GPU (vGPU).

`nvmlReturn_t nvmlDeviceGetPcieReplayCounter(nvmlDevice_t device, unsigned int *value)`

Parameters
- device
  The identifier of the target device
- value
  Reference in which to return the counter's value

Returns
- NVML_SUCCESS if value 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
Description
Retrieve the PCIe replay counter.
KEPLER_OR_NEWER%

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

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

Returns
- NVML_SUCCESS if clock has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clock is NULL
- NVML_ERROR_NOT_SUPPORTED if the device cannot report the specified clock
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the current clock speeds for the device.
FERMI_OR_NEWER%

See `nvmlClockType_t` for details on available clock information.

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

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

Returns
  ▶ NVML_SUCCESS if clock has been set
  ▶ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
  ▶ NVML_ERROR_INVALID_ARGUMENT if device is invalid or clock is NULL
  ▶ NVML_ERROR_NOT_SUPPORTED if the device cannot report the specified clock
  ▶ NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
  ▶ NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the maximum clock speeds for the device.

FERMI_OR_NEWER%

See `nvmlClockType_t` for details on available clock information.

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

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

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

Returns
  ▶ NVML_SUCCESS if clockMHz has been set
  ▶ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
Description
Retrieves the current setting of a clock that applications will use unless an overspec situation occurs. Can be changed using `nvmlDeviceSetApplicationsClocks`.

KEPLER_OR_NEWER%

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

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

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

Description
Retrieves the default applications clock that GPU boots with or defaults to after `nvmlDeviceResetApplicationsClocks` call.

KEPLER_OR_NEWER%
See also:

`nvmlDeviceGetApplicationsClock`

```c
nvmlReturn_t nvmlDeviceResetApplicationsClocks
   (nvmlDevice_t device)
```

**Parameters**

`device`

The identifier of the target device

**Returns**

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

**Description**

Resets the application clock to the default value

This is the applications clock that will be used after system reboot or driver reload. Default value is constant, but the current value can be changed using `nvmlDeviceSetApplicationsClocks`.

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

See also:

`nvmlDeviceGetApplicationsClock`

`nvmlDeviceSetApplicationsClocks`

FERMI_OR_NEWER_GF%
nvmlReturn_t nvmlDeviceGetClock (nvmlDevice_t device, nvmlClockType_t clockType, nvmlClockId_t clockId, unsigned int *clockMHz)

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

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

KEPLER_OR_NEWER%

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

Parameters

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

Returns
- NVML_SUCCESS if clockMHz has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clockMHz is NULL or clockType is invalid
- NVML_ERROR_NOT_SUPPORTED if the device 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

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

PASCAL_OR_NEWER%

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

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

Returns
- NVML_SUCCESS if count and clocksMHz have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or count is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_INSUFFICIENT_SIZE if count is too small (count is set to the number of required elements)
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

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

KEPLER_OR_NEWER%

**See also:**
nvmlDeviceSetApplicationsClocks
nvmlDeviceGetSupportedGraphicsClocks

```c
nvmlReturn_t nvmlDeviceGetSupportedGraphicsClocks
(nvmlDevice_t device, unsigned int memoryClockMHz,
unsigned int *count, unsigned int *clocksMHz)
```

**Parameters**
- `device` The identifier of the target device
- `memoryClockMHz` Memory clock for which to return possible graphics clocks
- `count` Reference in which to provide the `clocksMHz` array size, and to return the number of elements
- `clocksMHz` Reference in which to return the clocks in MHz

**Returns**
- NVML_SUCCESS if `count` and `clocksMHz` have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_FOUND if the specified `memoryClockMHz` is not a supported frequency
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or `clock` is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_INSUFFICIENT_SIZE if `count` is too small
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

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

KEPLER_OR_NEWER%

**See also:**

`nvmlDeviceSetApplicationsClocks`
`nvmlDeviceGetSupportedMemoryClocks`

```c
nvmlReturn_t nvmlDeviceGetAutoBoostedClocksEnabled (nvmlDevice_t device, nvmlEnableState_t *isEnabled, nvmlEnableState_t *defaultIsEnabled)
```

**Parameters**

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

**Returns**

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

**Description**

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

KEPLER_OR_NEWER%
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.

```c
nvmlReturn_t nvmlDeviceSetAutoBoostedClocksEnabled(nvmlDevice_t device, nvmlEnableState_t enabled)
```

**Parameters**

device
- The identifier of the target device

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

**Returns**

- NVML_SUCCESS If the Auto Boosted clocks were successfully set to the state specified by enabled
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support Auto Boosted clocks
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

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

KEPLER_OR_NEWER%

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.

```c
nvmlReturn_t
nvmlDeviceSetDefaultAutoBoostedClocksEnabled
(nvmlDevice_t device, nvmlEnableState_t enabled,
unsigned int flags)
```

**Parameters**

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

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

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

**Returns**

- NVML_SUCCESS If the Auto Boosted clock’s default state was successfully set to the state specified by enabled
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NO_PERMISSION If the calling user does not have permission to change Auto Boosted clock’s default state.
- NVML_ERROR_INVALID_ARGUMENT if device is invalid
- NVML_ERROR_NOT_SUPPORTED if the device does not support Auto Boosted clocks
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

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

KEPLER_OR_NEWER_GF% 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.

```c
nvmlReturn_t nvmlDeviceGetFanSpeed (nvmlDevice_t device, unsigned int *speed)
```

**Parameters**

- **device**
  - The identifier of the target device
- **speed**
  - Reference in which to return the fan speed percentage

**Returns**

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

**Description**

Retrieves the intended operating speed of the device’s fan.

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

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

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

**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

Description
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.

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

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

Parameters
device
The identifier of the target device
sensorType
Flag that indicates which sensor reading to retrieve
temp
Reference in which to return the temperature reading

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

**Description**

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

ALL_PRODUCTS%

See `nvmlTemperatureSensors_t` for details on available temperature sensors.

```c
nvmlReturn_t nvmlDeviceGetTemperatureThreshold (nvmlDevice_t device, nvmlTemperatureThresholds_t thresholdType, unsigned int *temp)
```

**Parameters**

- `device`
  The identifier of the target device
- `thresholdType`
  The type of threshold value queried
- `temp`
  Reference in which to return the temperature reading

**Returns**

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

**Description**

Retrieves the temperature threshold for the GPU with the specified threshold type in degrees C.
KEPLER_OR_NEWER%
See nvmlTemperatureThresholds_t for details on available temperature thresholds.

nvmlReturn_t nvmlDeviceGetPerformanceState
(nvmlDevice_t device, nvmlPstates_t *pState)

Parameters

device
The identifier of the target device

pState
Reference in which to return the performance state reading

Returns

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

Description
Retrieves the current performance state for the device.

FERMI_OR_NEWER%
See nvmlPstates_t for details on allowed performance states.

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

Parameters

device
The identifier of the target device

clocksThrottleReasons
Reference in which to return bitmask of active clocks throttle reasons
Returns

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

Description

Retrieves current clocks throttling reasons.

**FULL_SUPPORT%**

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

See also:

NvmlClocksThrottleReasons
nvmlDeviceGetSupportedClocksThrottleReasons

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

Parameters

- **device**
  The identifier of the target device
- **supportedClocksThrottleReasons**
  Reference in which to return bitmask of supported clocks throttle reasons

Returns

- NVML_SUCCESS if `supportedClocksThrottleReasons` has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or `supportedClocksThrottleReasons` is NULL
Description
Retrieves bitmask of supported clocks throttle reasons that can be returned by nvmlDeviceGetCurrentClocksThrottleReasons

FULL_SUPPORT%

This method is not supported in virtual machines running virtual GPU (vGPU).

See also:
NvmlClocksThrottleReasons
nvmlDeviceGetCurrentClocksThrottleReasons

nvmlReturn_t nvmlDeviceGetPowerState (nvmlDevice_t device, nvmlPstates_t *pState)

Parameters

device
   The identifier of the target device

pState
   Reference in which to return the performance state reading

Returns

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

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

Retrieve the current performance state for the device.

FERMI_OR_NEWER%
See `nvmlPstates_t` for details on allowed performance states.

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

**Parameters**

**device**

The identifier of the target device

**mode**

Reference in which to return the current power management mode

**Returns**

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

**Description**

This API has been deprecated.

Retrieves the power management mode associated with this device.

For products from the Fermi family.

- Requires `NVML_INFOROM_POWER` version 3.0 or higher.

For from the Kepler or newer families.

- Does not require `NVML_INFOROM_POWER` object.

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

See `nvmlEnableState_t` for details on allowed modes.
nvmlReturn_t nvmlDeviceGetPowerManagementLimit
(nvmlDevice_t device, unsigned int *limit)

Parameters

device
The identifier of the target device

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

Returns

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

Description
Retrieves the power management limit associated with this device.

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

This reading is only available if power management mode is supported. See nvmlDeviceGetPowerManagementMode.

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

Parameters

device
The identifier of the target device

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

Returns

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

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

KEPLER_OR_NEWER%

See also:

nvmlDeviceSetPowerManagementLimit

nvmlReturn_t
nvmlDeviceGetPowerManagementDefaultLimit (nvmlDevice_t device, unsigned int *defaultLimit)

Parameters

device
The identifier of the target device
defaultLimit
Reference in which to return the default power management limit in milliwatts

Returns

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

Description
Retrieves default power management limit on this device, in milliwatts. Default power management limit is a power management limit that the device boots with.

KEPLER_OR_NEWER%

`nvmlReturn_t nvmlDeviceGetPowerUsage (nvmlDevice_t device, unsigned int *power)`

Parameters

- **device**
  The identifier of the target device
- **power**
  Reference in which to return the power usage information

Returns

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

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

FERMI_OR_NEWER%

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

It is only available if power management mode is supported. See `nvmlDeviceGetPowerManagementMode`.
nvmlReturn_t nvmlDeviceGetTotalEnergyConsumption (nvmlDevice_t device, unsigned long long *energy)

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

Description

Retrieves total energy consumption for this GPU in millijoules (mJ) since the driver was last reloaded

VOLTA_OR_NEWER%

nvmlReturn_t nvmlDeviceGetEnforcedPowerLimit (nvmlDevice_t device, unsigned int *limit)

Parameters

device
   The device to communicate with
limit
   Reference in which to return the power management limit in milliwatts

Returns

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

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

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

KEPLER_OR_NEWER%

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

Parameters
device
The identifier of the target device
current
Reference in which to return the current GOM
pending
Reference in which to return the pending GOM

Returns
¬ NVML_SUCCESS if mode has been populated
¬ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
¬ NVML_ERROR_INVALID_ARGUMENT if device is invalid or current or pending is NULL
¬ NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
¬ NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
¬ NVML_ERROR_UNKNOWN on any unexpected error

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

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

nvmlGpuOperationMode_t
nvmlDeviceSetGpuOperationMode

nvmlReturn_t nvmlDeviceGetMemoryInfo (nvmlDevice_t device, nvmlMemory_t *memory)

Parameters

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

Returns

- NVML_SUCCESS if memory has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or memory is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the amount of used, free and total memory available on the device, in bytes.

ALL_PRODUCTS%

Enabling ECC reduces the amount of total available memory, due to the extra required parity bits. Under WDDM most device memory is allocated and managed on startup by Windows.

Under Linux and Windows TCC, the reported amount of used memory is equal to the sum of memory allocated by all active channels on the device.

See nvmlMemory_t for details on available memory info.
nvmlReturn_t nvmlDeviceGetComputeMode(nvmlDevice_t device, nvmlComputeMode_t *mode)

Parameters

device
   The identifier of the target device
mode
   Reference in which to return the current compute mode

Returns

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

Description

Retrieves the current compute mode for the device.

See nvmlComputeMode_t for details on allowed compute modes.

See also:

nvmlDeviceSetComputeMode()

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

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

Description

Retrieves the CUDA compute capability of the device.

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()`.

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

Parameters

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

Returns

- NVML_SUCCESS if current and pending have been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or either current or pending is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
nvmlReturn_t nvmlDeviceGetBoardId (nvmlDevice_t device, unsigned int *boardId)

Parameters

device
The identifier of the target device

boardId
Reference in which to return the device’s board ID

Returns

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

Description

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

FERMI_OR_NEWER%

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

**Parameters**

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

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

**Returns**

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

**Description**

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

FERMI_OR_NEWER%

```
nvmlReturn_t nvmlDeviceGetTotalEccErrors
(nvmlDevice_t device, nvmlMemoryErrorType_t errorType,
 nvmlEccCounterType_t counterType, unsigned long long *eccCounts)
```

**Parameters**

- **device**
  The identifier of the target device
errorType
  Flag that specifies the type of the errors.

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

eccCounts
  Reference in which to return the specified ECC errors

Returns
  • NVML_SUCCESS if eccCounts has been set
  • NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
  • NVML_ERROR_INVALID_ARGUMENT if device, errorType or counterType is invalid, or eccCounts is NULL
  • NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
  • NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
  • NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the total ECC error counts for the device.

FERMI_OR_NEWER% Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 1.0 or higher. Requires ECC Mode to be enabled.

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

See nvmlMemoryErrorType_t for a description of available error types. See nvmlEccCounterType_t for a description of available counter types.

See also:
nvmlDeviceClearEccErrorCounts()

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

Parameters
device
  The identifier of the target device
**errorType**
Flag that specifies the type of the errors.

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

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

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

**Description**
Retrieves the detailed ECC error counts for the device.

*Deprecated* This API supports only a fixed set of ECC error locations. On different GPU architectures, different locations are supported. See `nvmlDeviceGetMemoryErrorCounterFERMI_OR_NEWER`% 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()`

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

Parameters

device
The identifier of the target device

type
Flag that specifies the type of error.

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

locationType
Specifies the location of the counter.

count
Reference in which to return the ECC counter

Returns

- NVML_SUCCESS if count has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device, bitType, counterType or locationType is invalid, or count is NULL
- NVML_ERROR_NOT_SUPPORTED if the device does not support ECC error reporting in the specified memory
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieves the requested memory error counter for the device.

FERMI_OR_NEWER% Requires NVML_INFOROM_ECC version 2.0 or higher to report aggregate location-based memory error counts. Requires NVML_INFOROM_ECC version 1.0 or higher to report all other memory error counts.

Only applicable to devices with ECC.

Requires ECC Mode to be enabled.

See nvmlMemoryErrorType_t for a description of available memory error types. See nvmlEccCounterType_t for a description of available counter types. See nvmlMemoryLocation_t for a description of available counter locations.
nvmlReturn_t nvmlDeviceGetUtilizationRates(nvmlDevice_t device, nvmlUtilization_t *utilization)

Parameters

device
   The identifier of the target device
utilization
   Reference in which to return the utilization information

Returns

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

Description

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

FERMI_OR_NEWER%

See nvmlUtilization_t for details on available utilization rates.

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

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

Parameters

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

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

Description
Retrieves the current utilization and sampling size in microseconds for the Encoder KEPLER_OR_NEWER%

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

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 encoderQueryType
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
Description

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

MAXWELL_OR_NEWER%

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

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

Description

Retrieves the current encoder statistics for a given device.

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

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_UNKNOWN on any unexpected error

Description

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 accomodate 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.

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

Parameters

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

Returns

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

Description

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

nvmlReturn_t nvmlDeviceGetFBCStats (nvmlDevice_t device, nvmlFBCStats_t *fbcStats)

Parameters

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

Returns

▪ NVML_SUCCESS if fbcStats is fetched
Description
Retrieves the active frame buffer capture sessions statistics for a given device.

MAXWELL_OR_NEWER%

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

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

Description
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`.

MAXWELL_OR_NEWER%

```
| 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. |
```

```c
nvmlReturn_t nvmlDeviceGetDriverModel (nvmlDevice_t device, nvmlDriverModel_t *current, nvmlDriverModel_t *pending)
```

**Parameters**

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

**Returns**

- NVML_SUCCESS if either current and/or pending have been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or both current and pending are NULL
- NVML_ERROR_NOT_SUPPORTED if the platform is not windows
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

Retrieves the current and pending driver model for the device.

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

See `nvmlDriverModel_t` for details on available driver models.

**See also:**
`nvmlDeviceSetDriverModel()`

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

**Parameters**

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

- **version**
  
  Reference to which to return the VBIOS version

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

**Returns**

- NVML_SUCCESS if version has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, or version is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if length is too small
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

**Description**

Get VBIOS version of the device.

The VBIOS version may change from time to time. It will not exceed 32 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE.
nvmlReturn_t nvmlDeviceGetBridgeChipInfo
(nvmlDevice_t device, nvmlBridgeChipHierarchy_t
*bridgeHierarchy)

Parameters

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

Returns

‣ NVML_SUCCESS if bridge chip exists
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if device is invalid, or bridgeInfo is NULL
‣ NVML_ERROR_NOT_SUPPORTED if bridge chip not supported on the device
‣ NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description

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

FULL_SUPPORT% Only applicable to multi-GPU products.

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

Parameters

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

- NVML_SUCCESS if infoCount and infos have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if infoCount indicates that the infos array is too small infoCount will contain minimal amount of space necessary for the call to complete
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, either of infoCount or infos is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Get information about processes with a compute context on a device

FERMI_OR_NEWER%

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

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

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

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

See also:

nvmlSystemGetProcessName

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

Parameters

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

infos
Reference in which to return the process information

Returns
- NVML_SUCCESS if infoCount and infos have been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INSUFFICIENT_SIZE if infoCount indicates that the infos array is too small infoCount will contain minimal amount of space necessary for the call to complete
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, either of infoCount or infos is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

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

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

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

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

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

See also:
 nvmlSystemGetProcessName
nvmlReturn_t nvmlDeviceOnSameBoard (nvmlDevice_t device1, nvmlDevice_t device2, int *onSameBoard)

Parameters

device1
  The first GPU device
device2
  The second GPU device
onSameBoard
  Reference in which to return the status. Non-zero indicates that the GPUs are on the same board.

Returns

- NVML_SUCCESS if onSameBoard has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if dev1 or dev2 are invalid or onSameBoard is NULL
- NVML_ERROR_NOT_SUPPORTED if this check is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the either GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

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

FULL_SUPPORT%

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

Parameters

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

Reference in which to return the current restriction NVML_FEATURE_ENABLED indicates that the API is root-only NVML_FEATURE_DISABLED indicates that the API is accessible to all users

**Returns**

- NVML_SUCCESS if isRestricted has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, apiType incorrect or isRestricted is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device or the device does not support the feature that is being queried (E.G. Enabling/disabling Auto Boosted clocks is not supported by the device)
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERRORUNKNOWN on any unexpected error

**Description**

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

**See also:**

nvmlRestrictedAPI_t

---

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

**Parameters**

- **device**
  
  The identifier for the target device

- **type**
  
  Type of sampling event

- **lastSeenTimeStamp**
  
  Return only samples with timestamp greater than lastSeenTimeStamp.
sampleValType
Output parameter to represent the type of sample value as described in nvmlSampleVal_t

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

samples
Reference in which samples are returned

Returns
- NVML_SUCCESS if samples are successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, samplesCount is NULL or reference to sampleCount is 0 for non null samples
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_FOUND if sample entries are not found
- NVML_ERROR_UNKNOWN on any unexpected error

Description
Gets recent samples for the GPU.

KEPLER_OR_NEWER%

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

Power, Utilization and Clock samples are returned as type "unsigned int" for the union nvmlValue_t.

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

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

This method fetches the number of entries which can be accommodated in the provided samples array, and the reference samplesCount is updated to indicate how many samples were actually retrieved. The advantage of using this method for samples in contrast to polling via existing methods is to get get higher frequency data at lower polling cost.
nvmlReturn_t nvmlDeviceGetBAR1MemoryInfo(nvmlDevice_t device, nvmlBAR1Memory_t *bar1Memory)

Parameters

device
  The identifier of the target device

bar1Memory
  Reference in which BAR1 memory information is returned.

Returns

- NVML_SUCCESS if BAR1 memory is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, bar1Memory is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Gets Total, Available and Used size of BAR1 memory.

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

KEPLER_OR_NEWER%

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

Parameters

device
  The identifier of the target device

perfPolicyType
  Represents Performance policy which can trigger GPU throttling

violTime
  Reference to which violation time related information is returned
Returns

- NVML_SUCCESS if violation time is successfully retrieved
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid, perfPolicyType is invalid, or violTime is NULL
- NVML_ERROR_NOT_SUPPORTED if this query is not supported by the device
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

Description

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

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

Violation for thermal capping is not supported at this time.

KEPLER_OR_NEWER%

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

Parameters

- **device**
  - The identifier of the target device
- **cause**
  - Filter page addresses by cause of retirement
- ** pageCount**
  - Reference in which to provide the addresses buffer size, and to return the number of retired pages that match cause. Set to 0 to query the size without allocating an addresses buffer
- **addresses**
  - Buffer to write the page addresses into

Returns

- NVML_SUCCESS if pageCount was populated and addresses was filled
NVML_ERROR_INSUFFICIENT_SIZE if pageCount indicates the buffer is not large enough to store all the matching page addresses. pageCount is set to the needed size.

NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized

NVML_ERROR_INVALID_ARGUMENT if device is invalid, pageCount is NULL, cause is invalid, or addresses is NULL

NVML_ERROR_NOT_SUPPORTED if the device doesn't support this feature

NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible

NVML_ERROR_UNKNOWN on any unexpected error

Description

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

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

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

Description

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

nvmlDeviceGetRetiredPages_v2 adds an additional timestamps parameter to return the time of each page’s retirement.

KEPLER_OR_NEWER%

nvmlReturn_t nvmlDeviceGetRetiredPagesPendingStatus(nvmlDevice_t device, nvmlEnableState_t *isPending)

Parameters

device
  The identifier of the target device
isPending
  Reference in which to return the pending status

Returns

- NVML_SUCCESS if isPending was populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or isPending is NULL
- NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error
Description
Check if any pages are pending retirement and need a reboot to fully retire.
KEPLER_OR_NEWER%

4.19. Unit Commands

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

nvmlReturn_t nvmlUnitSetLedState (nvmlUnit_t unit, nvmlLedColor_t color)

Parameters
unit
  The identifier of the target unit
color
  The target LED color

Returns
‣ NVML_SUCCESS if the LED color has been set
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if unit or color is invalid
‣ NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
‣ NVML_ERROR_NO_PERMISSION if the user doesn’t have permission to perform this operation
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description
Set the LED state for the unit. The LED can be either green (0) or amber (1).
S_CLASS% 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.
4.20. 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.

\textbf{nvmlReturn_t nvmlDeviceSetPersistenceMode}
\textbf{(nvmlDevice_t device, nvmlEnableState_t mode)}

\textbf{Parameters}

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

\textbf{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

\textbf{Description}

Set the persistence mode for the device.

\texttt{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 \texttt{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()

nvmlReturn_t nvmlDeviceSetComputeMode
(nvmlDevice_t device, nvmlComputeMode_t mode)

Parameters

device
The identifier of the target device

mode
The target compute mode

Returns

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

Description

Set the compute mode for the device.

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()

nvmlReturn_t nvmlDeviceSetEccMode (nvmlDevice_t device, nvmlEnableState_t ecc)

Parameters

device
    The identifier of the target device

ecc
    The target ECC mode

Returns

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

Description

Set the ECC mode for the device.

KEPLER_OR_NEWER% Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 1.0 or higher. Requires root/admin permissions.

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

This operation takes effect after the next reboot.

See nvmlEnableState_t for details on available modes.

See also:

nvmlDeviceGetEccMode()
nvmlReturn_t nvmlDeviceClearEccErrorCounts
(nvmlDevice_t device, nvmlEccCounterType_t
counterType)

Parameters

device
The identifier of the target device
counterType
Flag that indicates which type of errors should be cleared.

Returns

‣ NVML_SUCCESS if the error counts were cleared
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if device is invalid or counterType is invalid
‣ NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
‣ NVML_ERROR_NO_PERMISSION if the user doesn't have permission to perform this operation
‣ NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
‣ NVML_ERROR_UNKNOWN on any unexpected error

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

KEPLER_OR_NEWER% Only applicable to devices with ECC. Requires NVML_INFOROM_ECC version 2.0 or higher to clear aggregate location-based ECC counts. Requires NVML_INFOROM_ECC version 1.0 or higher to clear all other ECC counts. Requires root/admin permissions. Requires ECC Mode to be enabled.

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

This operation takes effect immediately.

See nvmlMemoryErrorType_t for details on available counter types.

See also:
‣ nvmlDeviceGetDetailedEccErrors()
‣ nvmlDeviceGetTotalEccErrors()
nvmlReturn_t nvmlDeviceSetDriverModel (nvmlDevice_t device, nvmlDriverModel_t driverModel, unsigned int flags)

Parameters

device  
The identifier of the target device
driverModel  
The target driver model
flags  
Flags that change the default behavior

Returns

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

Description

Set the driver model for the device.

FERMI_OR_NEWER% For windows only. Requires root/admin permissions.

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

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

This operation takes effect after the next reboot.

Windows driver model may only be set to WDDM when running in DEFAULT compute mode.
Change driver model to WDDM is not supported when GPU doesn’t support graphics acceleration or will not support it after reboot. See `nvmlDeviceSetGpuOperationMode`.

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

See also:

`nvmlDeviceGetDriverModel()`

```c
nvmlReturn_t nvmlDeviceSetGpuLockedClocks(nvmlDevice_t device, unsigned int minGpuClockMHz, unsigned int maxGpuClockMHz)
```

**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

**Description**

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. Setting this will supercede application clock values and take effect regardless if a cuda app is running. See `/ref nvmlDeviceSetApplicationsClocks`.
Can be used as a setting to request constant performance.

Requires root/admin permissions.

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

VOLTA_OR_NEWER%

nvmlReturn_t nvmlDeviceResetGpuLockedClocks (nvmlDevice_t device)

Parameters

device
  The identifier of the target device

Returns

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

Description

Resets the 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, but the current values can be changed using
nvmlDeviceSetApplicationsClocks.

See also:

nvmlDeviceSetGpuLockedClocks

VOLTA_OR_NEWER%
nvmlReturn_t nvmlDeviceSetApplicationsClocks
(nvmlDevice_t device, unsigned int memClockMHz,
unsigned int graphicsClockMHz)

Parameters

device
The identifier of the target device

memClockMHz
Requested memory clock in MHz

graphicsClockMHz
Requested graphics clock in MHz

Returns

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

Description

Set clocks that applications will lock to.

Sets the clocks that compute and graphics applications will be running at. e.g. CUDA
driver requests these clocks during context creation which means this property defines
clocks at which CUDA applications will be running unless some overspec event occurs
(e.g. over power, over thermal or external HW brake).

Can be used as a setting to request constant performance.

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

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

KEPLER_OR_NEWER_GF% Requires root/admin permissions.
See `nvmlDeviceGetSupportedMemoryClocks` and `nvmlDeviceGetSupportedGraphicsClocks` for details on how to list available clocks combinations.

After system reboot or driver reload applications clocks go back to their default value. See `nvmlDeviceResetApplicationsClocks`.

```c
nvmlReturn_t nvmlDeviceSetPowerManagementLimit
(nvmlDevice_t device, unsigned int limit)
```

**Parameters**

- **device**
  - The identifier of the target device
- **limit**
  - Power management limit in milliwatts to set

**Returns**

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

**Description**

Set new power limit of this device.

KEPLER_OR_NEWER% Requires root/admin permissions.

See `nvmlDeviceGetPowerManagementLimitConstraints` to check the allowed ranges of values.

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

**See also:**

- `nvmlDeviceGetPowerManagementLimitConstraints`
- `nvmlDeviceGetPowerManagementDefaultLimit`
nvmlReturn_t nvmlDeviceSetGpuOperationMode(nvmlDevice_t device, nvmlGpuOperationMode_t mode)

Parameters

device
The identifier of the target device

mode
Target GOM

Returns

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

Description

Sets new GOM. See `nvmlGpuOperationMode_t` for details.

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

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

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

See also:

`nvmlGpuOperationMode_t`
`nvmlDeviceGetGpuOperationMode`
nvmlReturn_t nvmlDeviceSetAPIRestriction
(nvmlDevice_t device, nvmlRestrictedAPI_t apiType,
nvmlEnableState_t isRestricted)

Parameters

device
The identifier of the target device

apiType
Target API type for this operation

isRestricted
The target restriction

Returns

‣ NVML_SUCCESS if isRestricted has been set
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if device is invalid or apiType incorrect
‣ NVML_ERROR_NOT_SUPPORTED if the device does not support changing API restrictions or the device does not support the feature that api restrictions are being set for (E.G. Enabling/disabling auto boosted clocks is not supported by the device)
‣ NVML_ERROR_NO_PERMISISON if the user doesn't have permission to perform this operation
‣ NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description
Changes the root/admin 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.

KEPLER_OR_NEWER% Requires root/admin permissions.

See also:
nvmlRestrictedAPI_t
4.21. NvLink Methods

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

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

**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

**Description**

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

`PASCAL_OR_NEWER% nvmlReturn_t nvmlDeviceGetNvLinkVersion (nvmlDevice_t device, unsigned int link, unsigned int *version)`

**Parameters**

- **device**
  The identifier of the target device
link
   Specifies the NvLink link to be queried
version
   Requested NvLink version

Returns
▶ NVML_SUCCESS if version has been set
▶ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
▶ NVML_ERROR_INVALID_ARGUMENT if device or link is invalid or version is NULL
▶ NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
▶ NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieves the version of the device’s NvLink for the link specified

PASCAL_OR_NEWER%

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

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

Description
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.

PASCAL_OR_NEWER%

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

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

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

PASCAL_OR_NEWER%

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

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

Description

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

PASCAL_OR_NEWER%

nvmlReturn_t nvmlDeviceResetNvLinkErrorCounters (nvmlDevice_t device, unsigned int link)

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

**Description**

Resets all error counters to zero. Please refer to `nvmlNvLinkErrorCounter_t` for the list of error counters that are reset.

PASCAL_OR_NEWER%

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

**Parameters**

- **device**
  The identifier of the target device
- **link**
  Specifies the NvLink link to be queried
- **counter**
  Specifies the counter that should be set (0 or 1).
- **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

**Description**

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.
PASCAL_OR_NEWER%

nvmlReturn_t nvmlDeviceGetNvLinkUtilizationControl
(nvmlDevice_t device, unsigned int link, unsigned int
counter, nvmlNvLinkUtilizationControl_t *control)

Parameters

device
   The identifier of the target device
link
   Specifies the NvLink link to be queried
counter
   Specifies the counter that should be set (0 or 1).
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

Description

Get the NVLINK utilization counter control information for the specified counter, 0 or 1. Please refer to nvmlNvLinkUtilizationControl_t for the structure definition

PASCAL_OR_NEWER%

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

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

Description
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
PASCAL_OR_NEWER%

nvmReturn_t
nvmlDeviceFreezeNvLinkUtilizationCounter
(nvmlDevice_t device, unsigned int link, unsigned int
counter, nvmlEnableState_t freeze)

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

Description

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

PASCAL_OR_NEWER%

nvmlReturn_t nvmlDeviceResetNvLinkUtilizationCounter
(nvmlDevice_t device, unsigned int link, unsigned int counter)

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

Description

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

PASCAL_OR_NEWER%
4.22. Event Handling Methods

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

struct nvmlEventData_t

Event Types

typedef struct nvmlEventSet_st *nvmlEventSet_t

Handle to an event set

nvmlReturn_t nvmlEventSetCreate (nvmlEventSet_t *set)

Parameters

set

Reference in which to return the event handle

Returns

‣ NVML_SUCCESS if the event has been set
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if set is NULL
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description

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

FERMI_OR_NEWER%

See also:

nvmlEventSetFree
nvmlReturn_t nvmlDeviceRegisterEvents (nvmlDevice_t device, unsigned long long eventTypes, nvmlEventSet_t set)

Parameters

device
  The identifier of the target device

eventTypes
  Bitmask of Event Types to record

set
  Set to which add new event types

Returns

- NVML_SUCCESS if the event has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if eventTypes is invalid or set is NULL
- NVML_ERROR_NOT_SUPPORTED if the platform does not support this feature or some of requested event types
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

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

FERMI_OR_NEWER% Ecc events are available only on ECC enabled devices (see nvmlDeviceGetTotalEccErrors) Power capping events are available only on Power Management enabled devices (see nvmlDeviceGetPowerManagementMode)

For Linux only.

IMPORTANT: Operations on set are not thread safe

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

If function reports NVML_ERROR_UNKNOWN, event set is in undefined state and should be freed. If function reports NVML_ERROR_NOT_SUPPORTED, event set can still be used. None of the requested eventTypes are registered in that case.
See also:
Event Types
nvmlDeviceGetSupportedEventTypes
nvmlEventSetWait
nvmlEventSetFree

nvmlReturn_t nvmlDeviceGetSupportedEventTypes
(nvmlDevice_t device, unsigned long long *eventTypes)

Parameters

device
  The identifier of the target device
eventTypes
  Reference in which to return bitmask of supported events

Returns

- NVML_SUCCESS if the eventTypes has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if eventType is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Returns information about events supported on device

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

See also:
Event Types
nvmlDeviceRegisterEvents
nvmlReturn_t nvmlEventSetWait (nvmlEventSet_t set, nvmlEventData_t *data, unsigned int timeoutms)

Parameters

set
Reference to set of events to wait on

data
Reference in which to return event data

timeoutms
Maximum amount of wait time in milliseconds for registered event

>Returns

- NVML_SUCCESS if the data has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if data is NULL
- NVML_ERROR_TIMEOUT if no event arrived in specified timeout or interrupt arrived
- NVML_ERROR_GPU_IS_LOST if a GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Waits on events and delivers events

FERMI_OR_NEWER%

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

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

See also:

Event Types

nvmlDeviceRegisterEvents
nvmlReturn_t nvmlEventSetFree (nvmlEventSet_t set)

Parameters
set
   Reference to events to be released

Returns
‣ NVML_SUCCESS if the event has been successfully released
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description
Releases events in the set
FERMI_OR_NEWER%

See also:
nvmlDeviceRegisterEvents

4.22.1. Event Types

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

See nvmlDeviceRegisterEvents and nvmlDeviceGetSupportedEventTypes to check which devices support each event.

Types can be combined with bitwise or operator ’|’ when passed to nvmlDeviceRegisterEvents

#define nvmlEventTypeSingleBitEccError 0x0000000000000001LL
Event about single bit ECC errors.

A corrected texture memory error is not an ECC error, so it does not generate a single bit event
#define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL
Event about double bit ECC errors.

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

#define nvmlEventTypePState 0x0000000000000004LL
Event about PState changes.

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

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

#define nvmlEventTypeClock 0x0000000000000010LL
Event about clock changes.
Kepler only

#define nvmlEventTypeNone 0x0000000000000000LL
Mask with no events.

#define nvmlEventTypeAll (nvmlEventTypeNone \ | nvmlEventTypeSingleBitEccError \ | nvmlEventTypeDoubleBitEccError \ | nvmlEventTypePState \ | nvmlEventTypeClock \ | nvmlEventTypeXidCriticalError \ )
Mask of all events.

4.23. 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.
nvmlReturn_t nvmlDeviceModifyDrainState (nvmlPciInfo_t *pciInfo, nvmlEnableState_t newState)

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

Description
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.
PASCAL_OR_NEWER% Some Kepler devices supported.

nvmlReturn_t nvmlDeviceQueryDrainState (nvmlPciInfo_t *pciInfo, nvmlEnableState_t *currentState)

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

Description

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.

PASCAL_OR_NEWER% Some Kepler devices supported.

```
nvmlReturn_t nvmlDeviceRemoveGpu (nvmlPciInfo_t *pciInfo, nvmlDetachGpuState_t gpuState, nvmlPcieLinkState_t linkState)
```

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

Description

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.

PASCAL_OR_NEWER% Some Kepler devices supported.

\texttt{nvmlReturn\_t nvmlDeviceDiscoverGpus (nvmlPciInfo\_t *pciInfo)}

**Parameters**

\texttt{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

**Description**

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.

PASCAL_OR_NEWER% Some Kepler devices supported.

## 4.24. Field Value Queries

This chapter describes NVML operations that are associated with retrieving Field Values from NVML.

`nvmlReturn_t nvmlDeviceGetFieldValues (nvmlDevice_t device, int valuesCount, nvmlFieldValue_t *values)`

**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

**Description**

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.

## 4.25. Grid Queries

This chapter describes NVML operations that are associated with NVIDIA GRID products.
nvmlReturn_t nvmlDeviceGetVirtualizationMode(nvmlDevice_t device, nvmlGpuVirtualizationMode_t *pVirtualMode)

Parameters

device
Identifier of the target device

pVirtualMode
Reference to virtualization mode. One of NVML_GPU_VIRTUALIZATION_*?

Returns

‣ NVML_SUCCESS if pVirtualMode is fetched
‣ NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
‣ NVML_ERROR_INVALID_ARGUMENT if device is invalid or pVirtualMode is NULL
‣ NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description

This method is used to get the virtualization mode corresponding to the GPU.

KEPLER_OR_NEWER%

4.26. Grid Commands

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

nvmlReturn_t nvmlDeviceSetVirtualizationMode(nvmlDevice_t device, nvmlGpuVirtualizationMode_t virtualMode)

Parameters

device
Identifier of the target device
virtualMode
virtualization mode. One of NVML_GPU_VIRTUALIZATION_?

Returns

- NVML_SUCCESS if pVirtualMode is set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or pVirtualMode is NULL
- NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
- NVML_ERROR_NOT_SUPPORTED if setting of virtualization mode is not supported.
- NVML_ERROR_NO_PERMISSION if setting of virtualization mode is not allowed for this client.

Description
This method is used to set the virtualization mode corresponding to the GPU.

KEPLER_OR_NEWER%

4.27. vGPU Management

Set of APIs supporting GRID vGPU

nvmlReturn_t nvmlDeviceGetSupportedVgpus(nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuTypeId_t *vgpuTypelds)

Parameters

device
The identifier of the target device

vgpuCount
Pointer to caller-supplied array size, and returns number of vGPU types

vgpuTypelds
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_VGPU_ECC_NOT_SUPPORTED if ECC is enabled on the device
- NVML_ERROR_UNKNOWN on any unexpected error

Description

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.

```c
nvmlReturn_t nvmlDeviceGetCreatableVgpus(nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuTypeId_t *vgpuTypeIds)
```

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_VGPU_ECC_NOT_SUPPORTED if ECC is enabled on the device
NVML_ERROR_UNKNOWN on any unexpected error

Description
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 vgpuTypeId. 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 createable 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.

nvmlReturn_t nvmlVgpuTypeGetClass
(nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeClass,
unsigned int *size)

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
Description
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`.

KEPLER_OR_NEWER%

`nvmlReturn_t nvmlVgpuTypeGetName(nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeName, unsigned int *size)`

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

Description
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`.

KEPLER_OR_NEWER%
nvmlReturn_t nvmlVgpuTypeGetDeviceID
(nvmlVgpuTypeSid_t vgpuTypeSid, unsigned long long *deviceID, unsigned long long *subsystemID)

Parameters

vgpuTypeSid
Handle to vGPU type
deviceID
Device ID and vendor ID of the device contained in single 32 bit value
subsystemID
Subsytem ID and subsytem 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 vgpuTypeSid is invalid, or deviceId or subsystemID are NULL
‣ NVML_ERROR_UNKNOWN on any unexpected error

Description
Retrieve the device ID of a vGPU type.
KEPLER_OR_NEWER%

nvmlReturn_t nvmlVgpuTypeGetFramebufferSize
(nvmlVgpuTypeSid_t vgpuTypeSid, unsigned long long *fbSize)

Parameters

vgpuTypeSid
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

Description
Retrieve the vGPU framebuffer size in bytes.

KEPLER_OR_NEWER%

\[
\text{nvmlReturn_t nvmlVgpuTypeGetNumDisplayHeads (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *numDisplayHeads)}
\]

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

Description
Retrieve count of vGPU’s supported display heads.

KEPLER_OR_NEWER%

\[
\text{nvmlReturn_t nvmlVgpuTypeGetResolution (nvmlVgpuTypeId_t vgpuTypeId, unsigned int displayIndex, unsigned int *xdim, unsigned int *ydim)}
\]

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

Description

Retrieve vGPU display head’s maximum supported resolution.

KEPLER_OR_NEWER%

def nvmlVgpuTypeGetLicense(nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeLicenseString, unsigned int size):

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
Description

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`.

KEPLER_OR_NEWER%

```c
nvmlReturn_t nvmlVgpuTypeGetFrameRateLimit(nvmlVgpuTypeId_t vgpuTypeIds, unsigned int *frameRateLimit)
```

Parameters

- **vgpuTypeIds**
  - 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 vgpuTypeIds is invalid, or frameRateLimit is NULL
- NVML_ERROR_UNKNOWN on any unexpected error

Description

Retrieve the static frame rate limit value of the vGPU type

KEPLER_OR_NEWER%
nvmlReturn_t nvmlVgpuTypeGetMaxInstances(nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCount)

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 device for a 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

Description

Retrieve the maximum number of vGPU instances creatable on a device for a given vGPU type

KEPLER_OR_NEWER%

nvmlReturn_t nvmlVgpuTypeGetMaxInstancesPerVm(nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCountPerVm)

Parameters

vgpuTypeId
    Handle to vGPU type

vgpuInstanceCountPerVm
    Pointer to get the max number of vGPU instances supported per VM for a 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

Description

Retrieve the maximum number of vGPU instances supported per VM for given vGPU type

KEPLER_OR_NEWER%

\[
\text{nvmlReturn_t nvmlDeviceGetActiveVgpus (nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuInstance_t *vgpuInstances)}
\]

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

Description

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.

KEPLER_OR_NEWER%

```c
nvmlReturn_t nvmlVgpuInstanceGetVmID
(nvmlVgpuInstance_t vgpuInstance, char *vmId,
unsigned int size, nvmlVgpuVmIdType_t *vmIdType)
```

**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

**Description**

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 NULL 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.
nvmlReturn_t nvmlVgpuInstanceGetUUID
(nvmlVgpuInstance_t vgpuInstance, char *uuid, unsigned int size)

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

Description

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.

nvmlReturn_t nvmlVgpuInstanceGetVmDriverVersion
(nvmlVgpuInstance_t vgpuInstance, char *version, unsigned int length)

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

Description
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 "Unknown" 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.

KEPLER_OR_NEWER%

**nvmlReturn_t nvmlVgpuInstanceGetFbUsage**
*(nvmlVgpuInstance_t vgpuInstance, unsigned long long *fbUsage)*

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

**Description**

Retrieve the framebuffer usage in bytes.

Framebuffer usage is the amount of vGPU framebuffer memory that is currently in use by the VM.

KEPLER_OR_NEWER%

```c
nvmlReturn_t nvmlVgpuInstanceGetLicenseStatus (nvmlVgpuInstance_t vgpuInstance, unsigned int *licensed)
```

**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

**Description**

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.

KEPLER_OR_NEWER%
nvmlReturn_t nvmlVgpuInstanceGetType(nvmlVgpuInstance_t vgpuInstance, nvmlVgpuTypeId_t *vgpuTypeId)

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

Description

Retrieve the vGPU type of a vGPU instance.
Returns the vGPU type ID of vgpu assigned to the vGPU instance.
KEPLER_OR_NEWER%

nvmlReturn_t nvmlVgpuInstanceGetFrameRateLimit(nvmlVgpuInstance_t vgpuInstance, unsigned int *frameRateLimit)

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

Description
Retrieve the frame rate limit set for the vGPU instance.
Returns the value of the frame rate limit set for the vGPU instance

KEPLER_OR_NEWER%

nvmlReturn_t nvmlVgpuInstanceGetEncoderCapacity
(nvmlVgpuInstance_t vgpuInstance, unsigned int *encoderCapacity)

Parameters
vgpuInstance
Identifier of the target vGPU instance
derCategory
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

Description
Retrieve the encoder capacity of a vGPU instance, as a percentage of maximum encoder capacity with valid values in the range 0-100.

MAXWELL_OR_NEWER%
nvmlReturn_t nvmlVgpuInstanceSetEncoderCapacity
(nvmlVgpuInstance_t vgpuInstance, unsigned int
coderCapacity)

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

Description

Set the encoder capacity of a vGPU instance, as a percentage of maximum encoder
capacity with valid values in the range 0-100.

MAXWELL_OR_NEWER%

nvmlReturn_t nvmlDeviceGetVgpuUtilization
(nvmlDevice_t device, unsigned long long
lastSeenTimeStamp, nvmlValueType_t *sampleValType,
unsigned int *vgpuInstanceSamplesCount,
nvmlVgpuInstanceUtilizationSample_t
*utilizationSamples)

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

Description
Retrieves current utilization for vGPUs on a physical GPU (device).

KEPLER_OR_NEWER%

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 "unsigned 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.

```
nvmlReturn_t nvmlDeviceGetVgpuProcessUtilization
(nvmlDevice_t device, unsigned long
long lastSeenTimeStamp, unsigned
int *vgpuProcessSamplesCount,
nvmlVgpuProcessUtilizationSample_t
*utilizationSamples)
```

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

**Description**

Retrieves current utilization for processes running on vGPUs on a physical GPU (device).

MAXWELL_OR_NEWER%

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 time stamp retrieved from a previous query to read utilization since the previous query.

```c
nvmlReturn_t nvmlDeviceGetGridLicensableFeatures(nvmlDevice_t device, nvmlGridLicensableFeatures_t *pGridLicensableFeatures)
```

**Parameters**

- `device` Identifier of the target device
pGridLicensableFeatures
Pointer to structure in which GRID 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

Description
Retrieve the GRID licensable features.
Identifies whether the system supports GRID Software Licensing. If it does, return the list of licensable feature(s) and their current license status.
	nvmlReturn_t nvmlVgpuInstanceGetEncoderStats(nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, unsigned int *averageFps, unsigned int *averageLatency)

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
Description
Retrieves the current encoder statistics of a vGPU Instance

nvmlReturn_t nvmlVgpuInstanceGetEncoderSessions(nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, nvmlEncoderSessionInfo_t *sessionInfo)

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

Description
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 accomodate 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.

MAXWELL_OR_NEWER%
nvmlReturn_t nvmlVgpuInstanceGetFBCStats
(nvmlVgpuInstance_t vgpuInstance, nvmlFBCStats_t *fbcStats)

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

Description
Retrieves the active frame buffer capture sessions statistics of a vGPU Instance

MAXWELL_OR_NEWER%

nvmlReturn_t nvmlVgpuInstanceGetFBCSessions
(nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, nvmlFBCSessionInfo_t *sessionInfo)

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_ERRORUNKNOWN on any unexpected error

Description
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.

MAXWELL_OR_NEWER% 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.

```
nvmlReturn_t nvmlDeviceGetProcessUtilization(nvmlDevice_t device, nvmlProcessUtilizationSample_t *utilization, unsigned int *processSamplesCount, unsigned long long lastSeenTimeStamp)
```

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

Description
Retrieves the current utilization and process ID

MAXWELL_OR_NEWER%
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.

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.
nvmlReturn_t nvmlVgpuInstanceGetAccountingMode
(nvmlVgpuInstance_t vgpuInstance, nvmlEnableState_t *mode)

Parameters

vgpuInstance
  The identifier of the target vGPU VM

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_UNKNOWN on any unexpected error

Description

Queries the state of per process accounting mode on vGPU.

MAXWELL_OR_NEWER%

nvmlReturn_t nvmlVgpuInstanceGetAccountingPids
(nvmlVgpuInstance_t vgpuInstance, unsigned int *count,
unsigned int *pids)

Parameters

vgpuInstance
  The identifier of the target vGPU VM

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

Description

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.

MAXWELL_OR_NEWER%

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.

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

See also:

nvmlVgpuInstanceGetAccountingPids

nvmlReturn_t nvmlVgpuInstanceGetAccountingStats (nvmlVgpuInstance_t vgpuInstance, unsigned int pid, nvmlAccountingStats_t *stats)

Parameters

vgpuInstance
- The identifier of the target vGPU VM

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

Description
Queries process’s accounting stats.

MAXWELL_OR_NEWER%
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.

- Accounting Mode needs to be on. See nvmlVgpuInstanceGetAccountingMode.
- Only compute and graphics applications stats can be queried. Monitoring applications stats can’t be queried since they don’t contribute to GPU utilization.
- In case of pid collision stats of only the latest process (that terminated last) will be reported

4.28. vGPU Migration

This chapter describes NVML operations that are associated with vGPU Migration.
struct nvmlVgpuVersion_t
struct nvmlVgpuMetadata_t
struct nvmlVgpuPgpuMetadata_t
struct nvmlVgpuPgpuCompatibility_t
enum nvmlVgpuVmCompatibility_t

vGPU VM compatibility codes

Values

NVML_VGPU_VM_COMPATIBILITY_NONE = 0x0
vGPU is not runnable

NVML_VGPU_VM_COMPATIBILITY_COLD = 0x1
vGPU is runnable from a cold / powered-off state (ACPI S5)

NVML_VGPU_VM_COMPATIBILITY_HIBERNATE = 0x2
vGPU is runnable from a hibernated state (ACPI S4)

NVML_VGPU_VM_COMPATIBILITY_SLEEP = 0x4
vGPU is runnable from a sleeped state (ACPI S3)

NVML_VGPU_VM_COMPATIBILITY_LIVE = 0x8
vGPU is runnable from a live/paused (ACPI S0)

enum nvmlVgpuPgpuCompatibilityLimitCode_t

vGPU-pGPU compatibility limit codes

Values

NVML_VGPU_COMPATIBILITY_LIMIT_NONE = 0x0
Compatibility is not limited.

NVML_VGPU_COMPATIBILITY_LIMIT_HOST_DRIVER = 0x1
compatibility is limited by host driver version.

NVML_VGPU_COMPATIBILITY_LIMIT_GUEST_DRIVER = 0x2
Compatibility is limited by guest driver version.

NVML_VGPU_COMPATIBILITY_LIMIT_GPU = 0x4
Compatibility is limited by GPU hardware.

NVML_VGPU_COMPATIBILITY_LIMIT_OTHER = 0x80000000
Compatibility is limited by an undefined factor.
nvmlReturn_t nvmlVgpuInstanceGetMetadata(nvmlVgpuInstance_t vgpuInstance, nvmlVgpuMetadata_t *vgpuMetadata, unsigned int *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

Description

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 GRID 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.

```
nvmlReturn_t nvmlDeviceGetVgpuMetadata(nvmlDevice_t device, nvmlVgpuPgpuMetadata_t *pgpuMetadata, unsigned int *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

**Description**

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.

```
nvmlReturn_t nvmlGetVgpuCompatibility(nvmlVgpuMetadata_t *vgpuMetadata,
```

---

[www.nvidia.com](http://www.nvidia.com)  
NVML  
vr418 | 190
nvmlVgpuPgpuMetadata_t *pgpuMetadata,
nvmlVgpuPgpuCompatibility_t *compatibilityInfo)

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

Description

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 compatibility. (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.

nvmlReturn_t nvmlGetVgpuVersion (nvmlVgpuVersion_t
*supported, nvmlVgpuVersion_t *current)

Parameters

supported
Pointer to caller-supplied structure into which the supported vGPU version range is
returned
current
Pointer to caller-supplied structure into which the caller enforced supported vGPU version range is returned.

Returns
- NVML_SUCCESS vGPU version range structure was successfully returned
- NVML_ERROR_NOT_SUPPORTED API not supported
- NVML_ERROR_UNKNOWN Error while getting the data

Description
Returns the following two version range structures `nvmlVgpuVersion_t`:
1. supported: structure representing the range of vGPU versions supported by the host;
2. current: structure representing the range of supported versions enforced by the caller via `nvmlSetVgpuVersion()`.

The caller pass in the pointer to the structures, into which the compatible ranges are written.

: 1. The guest driver will fail to load if the version is below the range returned in the current structure. 2. If the guest driver is above the range, it will be downgraded to the current structure maximum version.

```
nvmlReturn_t nvmlSetVgpuVersion (nvmlVgpuVersion_t *vgpuVersion)
```

Parameters

vgpuVersion
Pointer to caller-supplied vGPU supported version range.

Returns
- NVML_SUCCESS vGPU metadata structure was successfully returned
- NVML_ERROR_NOT_SUPPORTED API not supported
- NVML_ERROR_IN_USE Range not set as VM is running on the host
- NVML_ERROR_INVALID_ARGUMENT Range being set is outside the range supported by host driver
Description

Takes a vGPU version range structure `nvmlVgpuVersion_t` and set the vGPU compatible version range to the one provided as input. The caller should call the `nvmlGetVgpuVersion()` to get the range of supported version by the host driver.

!: 1. The guest driver will fail to load if the version is below the range set via vgpuVersion structure. 2. If the guest driver is above the range, it will be downgraded to the vgpuVersion structure maximum version. 3. This will result error if there are VMs already active on the host or the supported range being set is outside the range supported by host driver.

4.29. GPU Blacklist Queries

This chapter describes NVML operations that are associated with blacklisted GPUs.

```c
struct nvmlBlacklistDeviceInfo_t

nvmlReturn_t nvmlGetBlacklistDeviceCount (unsigned int *deviceCount)
```

Parameters

- `deviceCount` Reference in which to return the number of blacklisted devices

Returns

- `NVML_SUCCESS` if `deviceCount` has been set
- `NVML_ERROR_INVALID_ARGUMENT` if `deviceCount` is NULL

Description

Retrieves the number of blacklisted GPU devices in the system.

ALL_PRODUCTS%
nvmlReturn_t nvmlGetBlacklistDeviceInfoByIndex
(unsigned int index, nvmlBlacklistDeviceInfo_t *info)

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

Description

Acquire the device information for a blacklisted device, based on its index.

Valid indices are derived from the deviceCount returned by
nvmlGetBlacklistDeviceCount(). For example, if deviceCount is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

See also:

nvmlGetBlacklistDeviceCount

4.30. NvmlClocksThrottleReasons

#define nvmlClocksThrottleReasonGpuIdle
0x0000000000000001LL

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

This limiter may be removed in a later release
```c
#define nvmlClocksThrottleReasonApplicationsClocksSetting 0x0000000000000002LL

GPU clocks are limited by current setting of applications clocks

See also:
nvmlDeviceSetApplicationsClocks
nvmlDeviceGetApplicationsClock

#define nvmlClocksThrottleReasonUserDefinedClocks
nvmlClocksThrottleReasonApplicationsClocksSetting

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

#define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL

SW Power Scaling algorithm is reducing the clocks below requested clocks

See also:
nvmlDeviceGetPowerUsage
nvmlDeviceSetPowerManagementLimit
nvmlDeviceGetPowerManagementLimit

#define nvmlClocksThrottleReasonHwSlowdown 0x0000000000000008LL

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

This is an indicator of:

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

See also:
NVML

nvmlDeviceGetTemperature
nvmlDeviceGetTemperatureThreshold
nvmlDeviceGetPowerUsage

#define nvmlClocksThrottleReasonSyncBoost
0x0000000000000010LL

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

#define nvmlClocksThrottleReasonSwThermalSlowdown
0x0000000000000020LL

SW Thermal Slowdown
This is an indicator of one or more of the following:

- Current GPU temperature above the GPU Max Operating Temperature
- Current memory temperature above the Memory Max Operating Temperature

#define nvmlClocksThrottleReasonHwThermalSlowdown
0x0000000000000040LL

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
nvmlDeviceGetTemperatureThreshold
nvmlDeviceGetPowerUsage
#define nvmlClocksThrottleReasonHwPowerBrakeSlowdown
0x0000000000000080LL

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`
- `nvmlDeviceGetTemperatureThreshold`
- `nvmlDeviceGetPowerUsage`

#define nvmlClocksThrottleReasonDisplayClockSetting
0x0000000000000100LL

GPU clocks are limited by current setting of Display clocks

See also:

- `bug 1997531`

#define nvmlClocksThrottleReasonNone
0x0000000000000000LL

Bit mask representing no clocks throttling.

Clocks are as high as possible.

#define nvmlClocksThrottleReasonAll
(nvmlClocksThrottleReasonNone \ |
 nvmlClocksThrottleReasonGpuIdle \ |
 nvmlClocksThrottleReasonApplicationsClocksSetting \ |
 nvmlClocksThrottleReasonSwPowerCap \ |
 nvmlClocksThrottleReasonHwSlowdown \ |
 nvmlClocksThrottleReasonSyncBoost \ |
 nvmlClocksThrottleReasonSwThermalSlowdown \ |
nvmlClocksThrottleReasonHwThermalSlowdown \ | 
nvmlClocksThrottleReasonHwPowerBrakeSlowdown \ | 
nvmlClocksThrottleReasonDisplayClockSetting \ )

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

Here are the data structures with brief descriptions:

- `nvmlAccountingStats_t`
- `nvmlBAR1Memory_t`
- `nvmlBlacklistDeviceInfo_t`
- `nvmlBridgeChipHierarchy_t`
- `nvmlBridgeChipInfo_t`
- `nvmlEccErrorCounts_t`
- `nvmlEncoderSessionInfo_t`
- `nvmlEventData_t`
- `nvmlFBCSessionInfo_t`
- `nvmlFBCStats_t`
- `nvmlFieldValue_t`
- `nvmlGridLicensableFeature_t`
- `nvmlGridLicensableFeatures_t`
- `nvmlHwbcEntry_t`
- `nvmlLedState_t`
- `nvmlMemory_t`
- `nvmlNvLinkUtilizationControl_t`
- `nvmlPciInfo_t`
- `nvmlProcessInfo_t`
- `nvmlProcessUtilizationSample_t`
- `nvmlPSUInfo_t`
- `nvmlSample_t`
- `nvmlUnitFanInfo_t`
- `nvmlUnitFanSpeeds_t`
- `nvmlUnitInfo_t`
- `nvmlUtilization_t`
- `nvmlValue_t`
- `nvmlVgpuInstanceUtilizationSample_t`
- `nvmlVgpuMetadata_t`
5.1. `nvmlAccountingStats_t` Struct Reference

Describes accounting statistics of a process.

### `unsigned int nvmlAccountingStats_t::gpuUtilization`

**Description**
Percent of time over the process's lifetime during which one or more kernels was executing on the GPU. Utilization stats just like returned by `nvmlDeviceGetUtilizationRates` but for the life time of a process (not just the last sample period). Set to NVML_VALUE_NOT_AVAILABLE if `nvmlDeviceGetUtilizationRates` is not supported.

### `unsigned int nvmlAccountingStats_t::memoryUtilization`

**Description**
Percent of time over the process's lifetime during which global (device) memory was being read or written. Set to NVML_VALUE_NOT_AVAILABLE if `nvmlDeviceGetUtilizationRates` is not supported.

### `unsigned long long nvmlAccountingStats_t::maxMemoryUsage`

**Description**
Maximum total memory in bytes that was ever allocated by the process. Set to NVML_VALUE_NOT_AVAILABLE if `nvmlProcessInfo_t->usedGpuMemory` is not supported.

### `unsigned long long nvmlAccountingStats_t::time`

**Description**
Amount of time in ms during which the compute context was active. The time is reported as 0 if the process is not terminated.
unsigned long long nvmlAccountingStats_t::startTime
CPU Timestamp in usec representing start time for the process.

unsigned int nvmlAccountingStats_t::isRunning
Flag to represent if the process is running (1 for running, 0 for terminated).

unsigned int nvmlAccountingStats_t::reserved
Reserved for future use.

5.2. nvmlBAR1Memory_t Struct Reference
BAR1 Memory allocation Information for a device

unsigned long long nvmlBAR1Memory_t::bar1Total
Total BAR1 Memory (in bytes).

unsigned long long nvmlBAR1Memory_t::bar1Free
Unallocated BAR1 Memory (in bytes).

unsigned long long nvmlBAR1Memory_t::bar1Used
Allocated Used Memory (in bytes).

5.3. nvmlBlacklistDeviceInfo_t Struct Reference
Blacklist GPU device information

struct nvmlPciInfo_t nvmlBlacklistDeviceInfo_t::pciInfo
The PCI information for the blacklisted GPU.

char nvmlBlacklistDeviceInfo_t::uuid
The ASCII string UUID for the blacklisted GPU.

5.4. nvmlBridgeChipHierarchy_t Struct Reference
This structure stores the complete Hierarchy of the Bridge Chip within the board. The immediate bridge is stored at index 0 of bridgeInfoList, parent to immediate bridge is at index 1 and so forth.
unsigned char nvmlBridgeChipHierarchy_t::bridgeCount
Number of Bridge Chips on the Board.

struct nvmlBridgeChipInfo_t
nvmlBridgeChipHierarchy_t::bridgeChipInfo
Hierarchy of Bridge Chips on the board.

5.5. nvmlBridgeChipInfo_t Struct Reference

Information about the Bridge Chip Firmware

nvmlBridgeChipType_t nvmlBridgeChipInfo_t::type
Type of Bridge Chip.

unsigned int nvmlBridgeChipInfo_t::fwVersion
Firmware Version. 0=Version is unavailable.

5.6. nvmlEccErrorCounts_t Struct Reference

Detailed ECC error counts for a device.
Deprecated Different GPU families can have different memory error counters See nvmlDeviceGetMemoryErrorCounter
unsigned long long nvmlEccErrorCounts_t::l1Cache
L1 cache errors.

unsigned long long nvmlEccErrorCounts_t::l2Cache
L2 cache errors.

unsigned long long nvmlEccErrorCounts_t::deviceMemory
Device memory errors.

unsigned long long nvmlEccErrorCounts_t::registerFile
Register file errors.

5.7. nvmlEncoderSessionInfo_t Struct Reference

Structure to hold encoder session data
unsigned int nvmlEncoderSessionInfo_t::sessionId
Unique session ID.

unsigned int nvmlEncoderSessionInfo_t::pid
Owning process ID.

nvmlVgpuInstance_t
nvmlEncoderSessionInfo_t::vgpuInstance
Owning vGPU instance ID (only valid on vGPU hosts, otherwise zero).

nvmlEncoderType_t
nvmlEncoderSessionInfo_t::codecType
Video encoder type.

unsigned int nvmlEncoderSessionInfo_t::hResolution
Current encode horizontal resolution.

unsigned int nvmlEncoderSessionInfo_t::vResolution
Current encode vertical resolution.

unsigned int nvmlEncoderSessionInfo_t::averageFps
Moving average encode frames per second.

unsigned int nvmlEncoderSessionInfo_t::averageLatency
Moving average encode latency in microseconds.

5.8. nvmlEventData_t Struct Reference
Information about occurred event
nvmlDevice_t nvmlEventData_t::device
Specific device where the event occurred.

unsigned long long nvmlEventData_t::eventType
Information about what specific event occurred.

unsigned long long nvmlEventData_t::eventData
Stores last XID error for the device in the event of nvmlEventTypeXidCriticalError.

5.9. nvmlFBCSessionInfo_t Struct Reference

Structure to hold FBC session data
unsigned int nvmlFBCSessionInfo_t::sessionId
Unique session ID.

unsigned int nvmlFBCSessionInfo_t::pid
Owning process ID.

nvmlVgpuInstance_t
nvmlFBCSessionInfo_t::vgpuInstance
Owning vGPU instance ID (only valid on vGPU hosts, otherwise zero).

unsigned int nvmlFBCSessionInfo_t::displayOrdinal
Display identifier.

nvmlFBCSessionType_t
nvmlFBCSessionInfo_t::sessionType
Type of frame buffer capture session.

unsigned int nvmlFBCSessionInfo_t::sessionFlags
Session flags (one or more of NVML_NVFBC_SESSION_FLAG_XXX).

unsigned int nvmlFBCSessionInfo_t::hMaxResolution
Max horizontal resolution supported by the capture session.

unsigned int nvmlFBCSessionInfo_t::vMaxResolution
Max vertical resolution supported by the capture session.

unsigned int nvmlFBCSessionInfo_t::hResolution
Horizontal resolution requested by caller in capture call.

unsigned int nvmlFBCSessionInfo_t::vResolution
Vertical resolution requested by caller in capture call.

unsigned int nvmlFBCSessionInfo_t::averageFPS
Moving average new frames captured per second.

unsigned int nvmlFBCSessionInfo_t::averageLatency
Moving average new frame capture latency in microseconds.
5.10. nvmlFBCStats_t Struct Reference

Structure to hold frame buffer capture sessions stats

unsigned int nvmlFBCStats_t::sessionsCount
Total no of sessions.

unsigned int nvmlFBCStats_t::averageFPS
Moving average new frames captured per second.

unsigned int nvmlFBCStats_t::averageLatency
Moving average new frame capture latency in microseconds.

5.11. nvmlFieldValue_t Struct Reference

Information for a Field Value Sample
unsigned int nvmlFieldValue_t::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 nvmlFieldValue_t::unused
Currently unused. This should be initialized to 0 by the caller before any API call.

long long nvmlFieldValue_t::timestamp
CPU Timestamp of this value in microseconds since 1970.

long long nvmlFieldValue_t::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 nvmlFieldValue_t::valueType
Type of the value stored in value.

nvmlReturn_t nvmlFieldValue_t::nvmlReturn
Return code for retrieving this value. This must be checked before looking at value, as value is undefined if nvmlReturn != NVML_SUCCESS.

nvmlFieldValue_t::value
Value for this field. This is only valid if nvmlReturn == NVML_SUCCESS.

5.12. nvmlGridLicensableFeature_t Struct

Reference
Structure containing GRID licensable feature information
nvmlGridLicenseFeatureCode_t
nvmlGridLicensableFeature_t::featureCode
Licensed feature code.

unsigned int nvmlGridLicensableFeature_t::featureState
Non-zero if feature is currently licensed, otherwise zero.

5.13. nvmlGridLicensableFeatures_t Struct Reference
Structure to store GRID licensable features

int
nvmlGridLicensableFeatures_t::isGridLicenseSupported
Non-zero if GRID Software Licensing is supported on the system, otherwise zero.

unsigned int
nvmlGridLicensableFeatures_t::licensableFeaturesCount
Entries returned in gridLicensableFeatures array.

struct nvmlGridLicensableFeature_t
nvmlGridLicensableFeatures_t::gridLicensableFeatures
Array of GRID licensable features.

5.14. nvmlHwbcEntry_t Struct Reference
Description of HWBC entry

5.15. nvmlLedState_t Struct Reference
LED states for an S-class unit.
char nvmlLedState_t::cause
If amber, a text description of the cause.

nvmlLedColor_t nvmlLedState_t::color
GREEN or AMBER.

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

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

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

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

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

5.18. nvmlPciInfo_t Struct Reference
PCI information about a GPU device.
char nvmlPciInfo_t::busIdLegacy
The legacy tuple domain:bus:device.function PCI identifier (& NULL terminator).

unsigned int nvmlPciInfo_t::domain
The PCI domain on which the device's bus resides, 0 to 0xffffffff.

unsigned int nvmlPciInfo_t::bus
The bus on which the device resides, 0 to 0xff.

unsigned int nvmlPciInfo_t::device
The device's id on the bus, 0 to 31.

unsigned int nvmlPciInfo_t::pciDeviceId
The combined 16-bit device id and 16-bit vendor id.

unsigned int nvmlPciInfo_t::pciSubSystemId
The 32-bit Sub System Device ID.

char nvmlPciInfo_t::busId
The tuple domain:bus:device.function PCI identifier (& NULL terminator).

5.19. nvmlProcessInfo_t Struct Reference
Information about running compute processes on the GPU

unsigned int nvmlProcessInfo_t::pid
Process ID.

unsigned long long nvmlProcessInfo_t::usedGpuMemory

Description
Amount of used GPU memory in bytes. Under WDDM, NVML_VALUE_NOT_AVAILABLE is always reported because Windows KMD manages all the memory and not the NVIDIA driver
5.20. nvmlProcessUtilizationSample_t Struct Reference

Structure to store utilization value and process Id

unsigned int nvmlProcessUtilizationSample_t::pid
PID of process.

unsigned long long nvmlProcessUtilizationSample_t::timeStamp
CPU Timestamp in microseconds.

unsigned int nvmlProcessUtilizationSample_t::smUtil
SM (3D/Compute) Util Value.

unsigned int nvmlProcessUtilizationSample_t::memUtil
Frame Buffer Memory Util Value.

unsigned int nvmlProcessUtilizationSample_t::encUtil
Encoder Util Value.

unsigned int nvmlProcessUtilizationSample_t::decUtil
Decoder Util Value.

5.21. nvmlPSUInfo_t Struct Reference

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

- High voltage
- Fan failure
- Heatsink temperature
- Current limit
- Voltage below UV alarm threshold
- Low-voltage
- SI2C remote off command
- MOD_DISABLE input
5.22. nvmlSample_t Struct Reference

Information for Sample

unsigned long long nvmlSample_t::timeStamp
CPU Timestamp in microseconds.

nvmlSample_t::sampleValue
Sample Value.

5.23. nvmlUnitFanInfo_t Struct Reference

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

unsigned int nvmlUnitFanInfo_t::speed
Fan speed (RPM).

nvmlFanState_t nvmlUnitFanInfo_t::state
Flag that indicates whether fan is working properly.

5.24. nvmlUnitFanSpeeds_t Struct Reference

Fan speed readings for an entire S-class unit.
struct nvmlUnitFanInfo_t nvmlUnitFanSpeeds_t::fans
Fan speed data for each fan.

unsigned int nvmlUnitFanSpeeds_t::count
Number of fans in unit.

5.25. nvmlUnitInfo_t Struct Reference
Static S-class unit info.

char nvmlUnitInfo_t::name
Product name.

char nvmlUnitInfo_t::id
Product identifier.

char nvmlUnitInfo_t::serial
Product serial number.

char nvmlUnitInfo_t::firmwareVersion
Firmware version.

5.26. nvmlUtilization_t Struct Reference
Utilization information for a device. Each sample period may be between 1 second and 1/6 second, depending on the product being queried.

unsigned int nvmlUtilization_t::gpu
Percent of time over the past sample period during which one or more kernels was executing on the GPU.

unsigned int nvmlUtilization_t::memory
Percent of time over the past sample period during which global (device) memory was being read or written.

5.27. nvmlValue_t Union Reference
Union to represent different types of Value
double nvmlValue_t::dVal
If the value is double.

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

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

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

signed long long nvmlValue_t::sllVal
If the value is signed long long.

5.28. nvmlVgpuInstanceUtilizationSample_t Struct
Reference

Structure to store Utilization Value and vgpuInstance
nvmlVgpuInstance_t
nvmlVgpuInstanceUtilizationSample_t::vgpuInstance
vGPU Instance

unsigned long long
nvmlVgpuInstanceUtilizationSample_t::timeStamp
CPU Timestamp in microseconds.

nvmlVgpuInstanceUtilizationSample_t::smUtil
SM (3D/Compute) Util Value.

nvmlVgpuInstanceUtilizationSample_t::memUtil
Frame Buffer Memory Util Value.

nvmlVgpuInstanceUtilizationSample_t::encUtil
Encoder Util Value.

nvmlVgpuInstanceUtilizationSample_t::decUtil
Decoder Util Value.

5.29. nvmlVgpuMetadata_t Struct Reference
vGPU metadata structure.
5.30. nvmlVgpuPgpuCompatibility_t Struct

Reference
vGPU-pGPU compatibility structure
nvmlVgpuVmCompatibility_t
nvmlVgpuPgpuCompatibility_t::vgpuVmCompatibility
Compatibility of vGPU VM. See nvmlVgpuVmCompatibility_t.

nvmlVgpuPgpuCompatibilityLimitCode_t
nvmlVgpuPgpuCompatibility_t::compatibilityLimitCode
Limiting factor for vGPU-pGPU compatibility. See
nvmlVgpuPgpuCompatibilityLimitCode_t.

5.31. nvmlVgpuPgpuMetadata_t Struct Reference
Physical GPU metadata structure
unsigned int nvmlVgpuPgpuMetadata_t::version
Current version of the structure.

unsigned int nvmlVgpuPgpuMetadata_t::revision
Current revision of the structure.

cchar nvmlVgpuPgpuMetadata_t::hostDriverVersion
Host driver version.

unsigned int
nvmlVgpuPgpuMetadata_t::pgpuVirtualizationCaps
Pgpu virtualization capabilities bitfileld.

unsigned int nvmlVgpuPgpuMetadata_t::reserved
Reserved for internal use.

struct nvmlVgpuVersion_t
nvmlVgpuPgpuMetadata_t::hostSupportedVgpuRange
vGPU version range supported by host driver

unsigned int nvmlVgpuPgpuMetadata_t::opaqueDataSize
Size of opaque data field in bytes.

cchar nvmlVgpuPgpuMetadata_t::opaqueData
Opaque data.

5.32. nvmlVgpuProcessUtilizationSample_t Struct Reference
Structure to store Utilization Value, vgpuInstance and subprocess information
nvmlVgpuInstance_t
nvmlVgpuProcessUtilizationSample_t::vgpuInstance
vGPU Instance

unsigned int nvmlVgpuProcessUtilizationSample_t::pid
PID of process running within the vGPU VM.

char nvmlVgpuProcessUtilizationSample_t::processName
Name of process running within the vGPU VM.

unsigned long long
nvmlVgpuProcessUtilizationSample_t::timeStamp
CPU Timestamp in microseconds.

unsigned int
nvmlVgpuProcessUtilizationSample_t::smUtil
SM (3D/Compute) Util Value.

unsigned int
nvmlVgpuProcessUtilizationSample_t::memUtil
Frame Buffer Memory Util Value.

unsigned int
nvmlVgpuProcessUtilizationSample_t::encUtil
Encoder Util Value.

unsigned int
nvmlVgpuProcessUtilizationSample_t::decUtil
Decoder Util Value.

5.33. nvmlVgpuVersion_t Struct Reference
Structure representing a range of vGPU version
unsigned int nvmlVgpuVersion_t::minVersion
Minimum vGPU version.

unsigned int nvmlVgpuVersion_t::maxVersion
Maximum vGPU version.

5.34. nvmlViolationTime_t Struct Reference
Struct to hold perf policy violation status data

unsigned long long nvmlViolationTime_t::referenceTime
referenceTime represents CPU timestamp in microseconds

unsigned long long nvmlViolationTime_t::violationTime
violationTime in Nanoseconds
Here is a list of all documented struct and union fields with links to the struct/union documentation for each field:

**A**
- **averageFps**
  - `nvmlEncoderSessionInfo_t`
- **averageFPS**
  - `nvmlFBCStats_t`
  - `nvmlFBCSessionInfo_t`
- **averageLatency**
  - `nvmlEncoderSessionInfo_t`
  - `nvmlFBCStats_t`
  - `nvmlFBCSessionInfo_t`

**B**
- **bar1Free**
  - `nvmlBAR1Memory_t`
- **bar1Total**
  - `nvmlBAR1Memory_t`
- **bar1Used**
  - `nvmlBAR1Memory_t`
- **bridgeChipInfo**
  - `nvmlBridgeChipHierarchy_t`
- **bridgeCount**
  - `nvmlBridgeChipHierarchy_t`
- **bus**
  - `nvmlPciInfo_t`
- **busId**
  - `nvmlPciInfo_t`
busIdLegacy
  nvmlPciInfo_t

C
  cause
    nvmlLedState_t
  codecType
    nvmlEncoderSessionInfo_t
  color
    nvmlLedState_t
  compatibilityLimitCode
    nvmlVgpuPgpuCompatibility_t
  count
    nvmlUnitFanSpeeds_t
  current
    nvmlPSUInfo_t

D
  decUtil
    nvmlVgpuInstanceUtilizationSample_t
    nvmlVgpuProcessUtilizationSample_t
    nvmlProcessUtilizationSample_t
  device
    nvmlPciInfo_t
    nvmlEventData_t
  deviceMemory
    nvmlEccErrorCounts_t
  displayOrdinal
    nvmlFBCSessionInfo_t
  domain
    nvmlPciInfo_t
  dVal
    nvmlValue_t

E
  encUtil
    nvmlVgpuInstanceUtilizationSample_t
    nvmlVgpuProcessUtilizationSample_t
    nvmlProcessUtilizationSample_t
  eventData
    nvmlEventData_t
  eventType
    nvmlEventData_t
Data Fields

F
fans
   nvmlUnitFanSpeeds_t
featureCode
   nvmlGridLicensableFeature_t
featureState
   nvmlGridLicensableFeature_t
fieldId
   nvmlFieldValue_t
firmwareVersion
   nvmlUnitInfo_t
free
   nvmlMemory_t
fwVersion
   nvmlBridgeChipInfo_t

G
gpu
   nvmlUtilization_t
gpuUtilization
   nvmlAccountingStats_t
gridLicensableFeatures
   nvmlGridLicensableFeatures_t
guestDriverVersion
   nvmlVgpuMetadata_t
guestInfoState
   nvmlVgpuMetadata_t
guestVgpuVersion
   nvmlVgpuMetadata_t

H
hMaxResolution
   nvmlFBCSessionInfo_t
hostDriverVersion
   nvmlVgpuMetadata_t
   nvmlVgpuPgpuMetadata_t
hostSupportedVgpuRange
   nvmlVgpuPgpuMetadata_t
hResolution
   nvmlEncoderSessionInfo_t
   nvmlFBCSessionInfo_t
Data Fields

id
  nvmlUnitInfo_t
isGridLicenseSupported
  nvmlGridLicensableFeatures_t
isRunning
  nvmlAccountingStats_t

l1Cache
  nvmlEccErrorCounts_t
l2Cache
  nvmlEccErrorCounts_t
latencyUsec
  nvmlFieldValue_t
licensableFeaturesCount
  nvmlGridLicensableFeatures_t

maxMemoryUsage
  nvmlAccountingStats_t
maxVersion
  nvmlVgpuVersion_t
memory
  nvmlUtilization_t
memoryUtilization
  nvmlAccountingStats_t
memUtil
  nvmlVgpuInstanceUtilizationSample_t
  nvmlVgpuProcessUtilizationSample_t
  nvmlProcessUtilizationSample_t
minVersion
  nvmlVgpuVersion_t

name
  nvmlUnitInfo_t
nvmlReturn
  nvmlFieldValue_t

opaqueData
  nvmlVgpuMetadata_t
Data Fields

opaqueDataSize
  nvmlVgpuMetadata_t
  nvmlVgpuPgpuMetadata_t

P
  pciDeviceId
    nvmlPciInfo_t
  pciInfo
    nvmlBlacklistDeviceInfo_t
  pciSubSystemId
    nvmlPciInfo_t
  pgpuVirtualizationCaps
    nvmlVgpuPgpuMetadata_t
  pid
    nvmlFBCSessionInfo_t
    nvmlProcessUtilizationSample_t
    nvmlEncoderSessionInfo_t
    nvmlProcessInfo_t
    nvmlVgpuProcessUtilizationSample_t
  power
    nvmlPSUInfo_t
  processName
    nvmlVgpuProcessUtilizationSample_t

R
  referenceTime
    nvmlViolationTime_t
  registerFile
    nvmlEccErrorCounts_t
  reserved
    nvmlVgpuMetadata_t
    nvmlVgpuPgpuMetadata_t
    nvmlAccountingStats_t
  revision
    nvmlVgpuPgpuMetadata_t
    nvmlVgpuMetadata_t

S
  sampleValue
    nvmlSample_t
  serial
    nvmlUnitInfo_t
Data Fields

- **sessionFlags**
  - `nvmlFBCSessionInfo_t`

- **sessionId**
  - `nvmlEncoderSessionInfo_t`
  - `nvmlFBCSessionInfo_t`

- **sessionsCount**
  - `nvmlFBCStats_t`

- **sessionType**
  - `nvmlFBCSessionInfo_t`

- **sllVal**
  - `nvmlValue_t`

- **smUtil**
  - `nvmlVgpuInstanceUtilizationSample_t`
  - `nvmlVgpuProcessUtilizationSample_t`
  - `nvmlProcessUtilizationSample_t`

- **speed**
  - `nvmlUnitFanInfo_t`

- **startTime**
  - `nvmlAccountingStats_t`

- **state**
  - `nvmlUnitFanInfo_t`
  - `nvmlPSUInfo_t`

- **T**
  - `nvmlAccountingStats_t`

- **time**
  - `nvmlAccountingStats_t`

- **timeStamp**
  - `nvmlSample_t`
  - `nvmlVgpuProcessUtilizationSample_t`

- **timestamp**
  - `nvmlFieldValue_t`

- **timeStamp**
  - `nvmlProcessUtilizationSample_t`
  - `nvmlVgpuInstanceUtilizationSample_t`

- **total**
  - `nvmlMemory_t`

- **type**
  - `nvmlBridgeChipInfo_t`

- **U**
  - `nvmlValue_t`
ullVal
  nvmlValue_t
ulVal
  nvmlValue_t
unused
  nvmlFieldValue_t
used
  nvmlMemory_t
usedGpuMemory
  nvmlProcessInfo_t
uuid
  nvmlBlacklistDeviceInfo_t

V
  value
    nvmlFieldValue_t
valueType
  nvmlFieldValue_t
version
  nvmlVgpuPgpuMetadata_t
  nvmlVgpuMetadata_t
vgpuInstance
  nvmlFBCSessionInfo_t
  nvmlVgpuInstanceUtilizationSample_t
  nvmlVgpuProcessUtilizationSample_t
  nvmlEncoderSessionInfo_t
vgpuVmCompatibility
  nvmlVgpuPgpuCompatibility_t
violationTime
  nvmlViolationTime_t
vMaxResolution
  nvmlFBCSessionInfo_t
voltage
  nvmlPSUInfo_t
vResolution
  nvmlFBCSessionInfo_t
  nvmlEncoderSessionInfo_t
Chapter 7.
DEPRECATED LIST

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

Global nvml EccBitType_t
See nvmlMemoryErrorType_t for a more flexible type

Global NVML_SINGLE_BIT_ECC
Mapped to NVML_MEMORY_ERROR_TYPE_CORRECTED

Global NVML_DOUBLE_BIT_ECC
Mapped to NVML_MEMORY_ERROR_TYPE_UNCORRECTED

Global nvmlDeviceGetHandleBySerial
Since more than one GPU can exist on a single board this function is deprecated in
favor of nvmlDeviceGetHandleByUUID. For dual GPU boards this function will
return NVML_ERROR_INVALID_ARGUMENT.

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

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

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

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

© 2007-2019 NVIDIA Corporation. All rights reserved.