



NVML API REFERENCE MANUAL

August 3, 2020

Version 450.64



Contents

1	Known issues in the current version of NVML library	1
2	Change log of NVML library	3
2.1	Changes between nvidia-smi v445 Update and v450 ===	4
2.2	Changes between nvidia-smi v418 Update and v445 ===	4
2.3	Changes between nvidia-smi v361 Update and v418	4
2.4	Changes between nvidia-smi v349 Update and v361	4
2.5	Changes between NVML v346 Update and v349	4
2.6	Changes between NVML v340 Update and v346	5
2.7	Changes between NVML v331 Update and v340	5
2.8	Changes between NVML v5.319 Update and v331	5
2.9	Changes between NVML v5.319 RC and v5.319 Update	6
2.10	Changes between NVML v4.304 and v5.319 RC	6
2.11	Changes between NVML v4.304 RC and v4.304 Production	6
2.12	Changes between NVML v3.295 and v4.304 RC	7
2.13	Changes between NVML v2.285 and v3.295	7
2.14	Changes between NVML v1.0 and v2.285	8
3	Deprecated List	9
4	Module Index	11
4.1	Modules	11
5	Data Structure Index	13
5.1	Data Structures	13
6	Module Documentation	15
6.1	Device Structs	15
6.1.1	Define Documentation	16
6.1.1.1	NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE	16

6.1.1.2	NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE	16
6.1.1.3	NVML_DEVICE_PCI_BUS_ID_FMT	16
6.1.1.4	NVML_DEVICE_PCI_BUS_ID_FMT_ARGS	16
6.1.1.5	NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT	17
6.1.1.6	NVML_MAX_PHYSICAL_BRIDGE	17
6.1.1.7	NVML_NVLINK_MAX_LINKS	17
6.1.1.8	NVML_VALUE_NOT_AVAILABLE	17
6.1.2	Enumeration Type Documentation	17
6.1.2.1	nvmlBridgeChipType_t	17
6.1.2.2	nvmlGpuTopologyLevel_t	17
6.1.2.3	nvmlNvLinkCapability_t	17
6.1.2.4	nvmlNvLinkErrorCounter_t	17
6.1.2.5	nvmlNvLinkUtilizationCountPktTypes_t	17
6.1.2.6	nvmlNvLinkUtilizationCountUnits_t	17
6.1.2.7	nvmlPcieUtilCounter_t	18
6.1.2.8	nvmlPerfPolicyType_t	18
6.1.2.9	nvmlSamplingType_t	18
6.1.2.10	nvmlValueType_t	18
6.2	Device Enums	19
6.2.1	Define Documentation	21
6.2.1.1	NVML_DOUBLE_BIT_ECC	21
6.2.1.2	NVML_SINGLE_BIT_ECC	22
6.2.1.3	nvmlEccBitType_t	22
6.2.2	Enumeration Type Documentation	22
6.2.2.1	nvmlBrandType_t	22
6.2.2.2	nvmlClockId_t	22
6.2.2.3	nvmlClockType_t	22
6.2.2.4	nvmlComputeMode_t	23
6.2.2.5	nvmlDriverModel_t	23
6.2.2.6	nvmlEccCounterType_t	23
6.2.2.7	nvmlEnableState_t	23
6.2.2.8	nvmlGpuOperationMode_t	24
6.2.2.9	nvmlInforomObject_t	24
6.2.2.10	nvmlMemoryErrorType_t	24
6.2.2.11	nvmlMemoryLocation_t	24
6.2.2.12	nvmlPageRetirementCause_t	25
6.2.2.13	nvmlPstates_t	25

6.2.2.14	nvmlRestrictedAPI_t	26
6.2.2.15	nvmlReturn_t	26
6.2.2.16	nvmlTemperatureSensors_t	27
6.2.2.17	nvmlTemperatureThresholds_t	27
6.3	GRID Virtualization Enums	28
6.3.1	Enumeration Type Documentation	28
6.3.1.1	nvmlGpuVirtualizationMode_t	28
6.3.1.2	nvmlGridLicenseFeatureCode_t	28
6.3.1.3	nvmlHostVgpuMode_t	29
6.3.1.4	nvmlVgpuGuestInfoState_t	29
6.3.1.5	nvmlVgpuVmIdType_t	29
6.4	GRID Virtualization Constants	30
6.4.1	Define Documentation	30
6.4.1.1	NVML_GRID_LICENSE_BUFFER_SIZE	30
6.4.1.2	NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION	30
6.4.1.3	NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION	30
6.5	GRID Virtualization Structs	31
6.5.1	Define Documentation	31
6.5.1.1	NVML_DEVICE_ARCH_KEPLER	31
6.6	Field Value Enums	32
6.6.1	Define Documentation	41
6.6.1.1	NVML_FI_DEV_ECC_CURRENT	41
6.6.1.2	NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID	41
6.6.1.3	NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_TX	41
6.7	Unit Structs	42
6.7.1	Enumeration Type Documentation	42
6.7.1.1	nvmlFanState_t	42
6.7.1.2	nvmlLedColor_t	42
6.8	Event Types	43
6.8.1	Detailed Description	43
6.8.2	Define Documentation	43
6.8.2.1	nvmlEventTypeClock	43
6.8.2.2	nvmlEventTypeDoubleBitEccError	43
6.8.2.3	nvmlEventTypePState	44
6.8.2.4	nvmlEventTypeSingleBitEccError	44
6.9	Accounting Statistics	45
6.9.1	Detailed Description	45

6.9.2	Function Documentation	45
6.9.2.1	nvmlDeviceClearAccountingPids	45
6.9.2.2	nvmlDeviceGetAccountingBufferSize	46
6.9.2.3	nvmlDeviceGetAccountingMode	46
6.9.2.4	nvmlDeviceGetAccountingPids	47
6.9.2.5	nvmlDeviceGetAccountingStats	47
6.9.2.6	nvmlDeviceSetAccountingMode	48
6.10	Encoder Structs	49
6.10.1	Enumeration Type Documentation	49
6.10.1.1	nvmlEncoderType_t	49
6.11	Frame Buffer Capture Structures	50
6.11.1	Enumeration Type Documentation	50
6.11.1.1	nvmlFBSessionType_t	50
6.12	definitions related to the drain state	51
6.12.1	Enumeration Type Documentation	51
6.12.1.1	nvmlDetachGpuState_t	51
6.12.1.2	nvmlPcieLinkState_t	51
6.13	Initialization and Cleanup	52
6.13.1	Detailed Description	52
6.13.2	Function Documentation	52
6.13.2.1	nvmlInit_v2	52
6.13.2.2	nvmlInitWithFlags	53
6.13.2.3	nvmlShutdown	53
6.14	Error reporting	54
6.14.1	Detailed Description	54
6.14.2	Function Documentation	54
6.14.2.1	nvmlErrorString	54
6.15	Constants	55
6.15.1	Define Documentation	55
6.15.1.1	NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE	55
6.15.1.2	NVML_DEVICE_NAME_BUFFER_SIZE	55
6.15.1.3	NVML_DEVICE_NAME_V2_BUFFER_SIZE	55
6.15.1.4	NVML_DEVICE_PART_NUMBER_BUFFER_SIZE	55
6.15.1.5	NVML_DEVICE_SERIAL_BUFFER_SIZE	55
6.15.1.6	NVML_DEVICE_UUID_BUFFER_SIZE	55
6.15.1.7	NVML_DEVICE_UUID_V2_BUFFER_SIZE	55
6.15.1.8	NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE	56

6.15.1.9	NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE	56
6.15.1.10	NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE	56
6.16	System Queries	57
6.16.1	Detailed Description	57
6.16.2	Define Documentation	57
6.16.2.1	NVML_CUDA_DRIVER_VERSION_MAJOR	57
6.16.3	Function Documentation	57
6.16.3.1	nvmlSystemGetCudaDriverVersion	57
6.16.3.2	nvmlSystemGetCudaDriverVersion_v2	57
6.16.3.3	nvmlSystemGetDriverVersion	58
6.16.3.4	nvmlSystemGetNVMLVersion	58
6.16.3.5	nvmlSystemGetProcessName	59
6.17	Unit Queries	60
6.17.1	Detailed Description	60
6.17.2	Function Documentation	60
6.17.2.1	nvmlSystemGetHicVersion	60
6.17.2.2	nvmlUnitGetCount	60
6.17.2.3	nvmlUnitGetDevices	61
6.17.2.4	nvmlUnitGetFanSpeedInfo	61
6.17.2.5	nvmlUnitGetHandleByIndex	62
6.17.2.6	nvmlUnitGetLedState	62
6.17.2.7	nvmlUnitGetPsuInfo	62
6.17.2.8	nvmlUnitGetTemperature	63
6.17.2.9	nvmlUnitGetUnitInfo	63
6.18	Device Queries	64
6.18.1	Detailed Description	67
6.18.2	Function Documentation	67
6.18.2.1	nvmlDeviceGetAPIRestriction	67
6.18.2.2	nvmlDeviceGetApplicationsClock	68
6.18.2.3	nvmlDeviceGetArchitecture	68
6.18.2.4	nvmlDeviceGetAttributes_v2	68
6.18.2.5	nvmlDeviceGetAutoBoostedClocksEnabled	69
6.18.2.6	nvmlDeviceGetBAR1MemoryInfo	69
6.18.2.7	nvmlDeviceGetBoardId	70
6.18.2.8	nvmlDeviceGetBoardPartNumber	70
6.18.2.9	nvmlDeviceGetBrand	71
6.18.2.10	nvmlDeviceGetBridgeChipInfo	71

6.18.2.11 nvmlDeviceGetClock	72
6.18.2.12 nvmlDeviceGetClockInfo	72
6.18.2.13 nvmlDeviceGetComputeMode	73
6.18.2.14 nvmlDeviceGetComputeRunningProcesses_v2	73
6.18.2.15 nvmlDeviceGetCount_v2	74
6.18.2.16 nvmlDeviceGetCudaComputeCapability	74
6.18.2.17 nvmlDeviceGetCurrentClocksThrottleReasons	75
6.18.2.18 nvmlDeviceGetCurrPcieLinkGeneration	75
6.18.2.19 nvmlDeviceGetCurrPcieLinkWidth	76
6.18.2.20 nvmlDeviceGetDecoderUtilization	76
6.18.2.21 nvmlDeviceGetDefaultApplicationsClock	77
6.18.2.22 nvmlDeviceGetDetailedEccErrors	77
6.18.2.23 nvmlDeviceGetDisplayActive	78
6.18.2.24 nvmlDeviceGetDisplayMode	79
6.18.2.25 nvmlDeviceGetDriverModel	79
6.18.2.26 nvmlDeviceGetEccMode	80
6.18.2.27 nvmlDeviceGetEncoderCapacity	80
6.18.2.28 nvmlDeviceGetEncoderSessions	81
6.18.2.29 nvmlDeviceGetEncoderStats	81
6.18.2.30 nvmlDeviceGetEncoderUtilization	82
6.18.2.31 nvmlDeviceGetEnforcedPowerLimit	82
6.18.2.32 nvmlDeviceGetFanSpeed	83
6.18.2.33 nvmlDeviceGetFanSpeed_v2	83
6.18.2.34 nvmlDeviceGetFBCSessions	84
6.18.2.35 nvmlDeviceGetFBCStats	85
6.18.2.36 nvmlDeviceGetGpuOperationMode	85
6.18.2.37 nvmlDeviceGetGraphicsRunningProcesses_v2	86
6.18.2.38 nvmlDeviceGetHandleByIndex_v2	86
6.18.2.39 nvmlDeviceGetHandleByPciBusId_v2	87
6.18.2.40 nvmlDeviceGetHandleBySerial	88
6.18.2.41 nvmlDeviceGetHandleByUUID	89
6.18.2.42 nvmlDeviceGetIndex	89
6.18.2.43 nvmlDeviceGetInforomConfigurationChecksum	90
6.18.2.44 nvmlDeviceGetInforomImageVersion	91
6.18.2.45 nvmlDeviceGetInforomVersion	91
6.18.2.46 nvmlDeviceGetMaxClockInfo	92
6.18.2.47 nvmlDeviceGetMaxCustomerBoostClock	92

6.18.2.48 nvmlDeviceGetMaxPcieLinkGeneration	93
6.18.2.49 nvmlDeviceGetMaxPcieLinkWidth	93
6.18.2.50 nvmlDeviceGetMemoryErrorCounter	94
6.18.2.51 nvmlDeviceGetMemoryInfo	95
6.18.2.52 nvmlDeviceGetMinorNumber	95
6.18.2.53 nvmlDeviceGetMultiGpuBoard	96
6.18.2.54 nvmlDeviceGetName	96
6.18.2.55 nvmlDeviceGetP2PStatus	97
6.18.2.56 nvmlDeviceGetPcieReplayCounter	97
6.18.2.57 nvmlDeviceGetPcieThroughput	97
6.18.2.58 nvmlDeviceGetPciInfo_v3	98
6.18.2.59 nvmlDeviceGetPerformanceState	98
6.18.2.60 nvmlDeviceGetPersistenceMode	99
6.18.2.61 nvmlDeviceGetPowerManagementDefaultLimit	99
6.18.2.62 nvmlDeviceGetPowerManagementLimit	100
6.18.2.63 nvmlDeviceGetPowerManagementLimitConstraints	100
6.18.2.64 nvmlDeviceGetPowerManagementMode	101
6.18.2.65 nvmlDeviceGetPowerState	101
6.18.2.66 nvmlDeviceGetPowerUsage	102
6.18.2.67 nvmlDeviceGetRemappedRows	102
6.18.2.68 nvmlDeviceGetRetiredPages	103
6.18.2.69 nvmlDeviceGetRetiredPages_v2	104
6.18.2.70 nvmlDeviceGetRetiredPagesPendingStatus	104
6.18.2.71 nvmlDeviceGetRowRemapperHistogram	105
6.18.2.72 nvmlDeviceGetSamples	105
6.18.2.73 nvmlDeviceGetSerial	106
6.18.2.74 nvmlDeviceGetSupportedClocksThrottleReasons	107
6.18.2.75 nvmlDeviceGetSupportedGraphicsClocks	107
6.18.2.76 nvmlDeviceGetSupportedMemoryClocks	108
6.18.2.77 nvmlDeviceGetTemperature	108
6.18.2.78 nvmlDeviceGetTemperatureThreshold	109
6.18.2.79 nvmlDeviceGetTopologyCommonAncestor	109
6.18.2.80 nvmlDeviceGetTopologyNearestGpus	110
6.18.2.81 nvmlDeviceGetTotalEccErrors	110
6.18.2.82 nvmlDeviceGetTotalEnergyConsumption	111
6.18.2.83 nvmlDeviceGetUtilizationRates	111
6.18.2.84 nvmlDeviceGetUUID	112

6.18.2.85	nvmlDeviceGetVbiosVersion	112
6.18.2.86	nvmlDeviceGetViolationStatus	113
6.18.2.87	nvmlDeviceOnSameBoard	113
6.18.2.88	nvmlDeviceResetApplicationsClocks	114
6.18.2.89	nvmlDeviceSetAutoBoostedClocksEnabled	114
6.18.2.90	nvmlDeviceSetDefaultAutoBoostedClocksEnabled	115
6.18.2.91	nvmlDeviceValidateInforom	116
6.18.2.92	nvmlSystemGetTopologyGpuSet	116
6.18.2.93	nvmlVgpuInstanceGetMdevUUID	116
6.19	CPU and Memory Affinity	118
6.19.1	Detailed Description	118
6.19.2	Function Documentation	118
6.19.2.1	nvmlDeviceClearCpuAffinity	118
6.19.2.2	nvmlDeviceGetCpuAffinity	119
6.19.2.3	nvmlDeviceGetCpuAffinityWithinScope	119
6.19.2.4	nvmlDeviceGetMemoryAffinity	120
6.19.2.5	nvmlDeviceSetCpuAffinity	120
6.20	Unit Commands	122
6.20.1	Detailed Description	122
6.20.2	Function Documentation	122
6.20.2.1	nvmlUnitSetLedState	122
6.21	Device Commands	123
6.21.1	Detailed Description	123
6.21.2	Function Documentation	123
6.21.2.1	nvmlDeviceClearEccErrorCounts	123
6.21.2.2	nvmlDeviceResetGpuLockedClocks	124
6.21.2.3	nvmlDeviceSetAPIRestriction	124
6.21.2.4	nvmlDeviceSetApplicationsClocks	125
6.21.2.5	nvmlDeviceSetComputeMode	126
6.21.2.6	nvmlDeviceSetDriverModel	127
6.21.2.7	nvmlDeviceSetEccMode	127
6.21.2.8	nvmlDeviceSetGpuLockedClocks	128
6.21.2.9	nvmlDeviceSetGpuOperationMode	129
6.21.2.10	nvmlDeviceSetPersistenceMode	129
6.21.2.11	nvmlDeviceSetPowerManagementLimit	130
6.22	NvLink Methods	132
6.22.1	Detailed Description	132

6.22.2	Function Documentation	132
6.22.2.1	nvmlDeviceFreezeNvLinkUtilizationCounter	132
6.22.2.2	nvmlDeviceGetNvLinkCapability	133
6.22.2.3	nvmlDeviceGetNvLinkErrorCounter	133
6.22.2.4	nvmlDeviceGetNvLinkRemotePciInfo_v2	134
6.22.2.5	nvmlDeviceGetNvLinkState	134
6.22.2.6	nvmlDeviceGetNvLinkUtilizationControl	134
6.22.2.7	nvmlDeviceGetNvLinkUtilizationCounter	135
6.22.2.8	nvmlDeviceGetNvLinkVersion	135
6.22.2.9	nvmlDeviceResetNvLinkErrorCounters	136
6.22.2.10	nvmlDeviceResetNvLinkUtilizationCounter	136
6.22.2.11	nvmlDeviceSetNvLinkUtilizationControl	137
6.23	Event Handling Methods	138
6.23.1	Detailed Description	138
6.23.2	Typedef Documentation	138
6.23.2.1	nvmlEventSet_t	138
6.23.3	Function Documentation	138
6.23.3.1	nvmlDeviceGetSupportedEventTypes	138
6.23.3.2	nvmlDeviceRegisterEvents	139
6.23.3.3	nvmlEventSetCreate	140
6.23.3.4	nvmlEventSetFree	140
6.23.3.5	nvmlEventSetWait_v2	140
6.24	Drain states	142
6.24.1	Detailed Description	142
6.24.2	Function Documentation	142
6.24.2.1	nvmlDeviceDiscoverGpus	142
6.24.2.2	nvmlDeviceModifyDrainState	143
6.24.2.3	nvmlDeviceQueryDrainState	143
6.24.2.4	nvmlDeviceRemoveGpu_v2	143
6.25	Field Value Queries	145
6.25.1	Detailed Description	145
6.25.2	Function Documentation	145
6.25.2.1	nvmlDeviceGetFieldValues	145
6.26	GRID Virtualization Enums, Constants and Structs	146
6.27	GRID Virtualization APIs	147
6.27.1	Detailed Description	147
6.27.2	Function Documentation	147

6.27.2.1	nvmlDeviceGetGridLicensableFeatures_v3	147
6.27.2.2	nvmlDeviceGetHostVgpuMode	147
6.27.2.3	nvmlDeviceGetProcessUtilization	148
6.27.2.4	nvmlDeviceGetVirtualizationMode	149
6.27.2.5	nvmlDeviceSetVirtualizationMode	149
6.28	GRID vGPU Management	150
6.28.1	Detailed Description	151
6.28.2	Function Documentation	151
6.28.2.1	nvmlDeviceGetActiveVgpus	151
6.28.2.2	nvmlDeviceGetCreatableVgpus	151
6.28.2.3	nvmlDeviceGetSupportedVgpus	152
6.28.2.4	nvmlVgpuInstanceGetEccMode	153
6.28.2.5	nvmlVgpuInstanceGetEncoderCapacity	153
6.28.2.6	nvmlVgpuInstanceGetEncoderSessions	153
6.28.2.7	nvmlVgpuInstanceGetEncoderStats	154
6.28.2.8	nvmlVgpuInstanceGetFBCSessions	154
6.28.2.9	nvmlVgpuInstanceGetFBCStats	155
6.28.2.10	nvmlVgpuInstanceGetFbUsage	156
6.28.2.11	nvmlVgpuInstanceGetFrameRateLimit	156
6.28.2.12	nvmlVgpuInstanceGetLicenseStatus	156
6.28.2.13	nvmlVgpuInstanceGetType	157
6.28.2.14	nvmlVgpuInstanceGetUUID	157
6.28.2.15	nvmlVgpuInstanceGetVmDriverVersion	158
6.28.2.16	nvmlVgpuInstanceGetVmID	158
6.28.2.17	nvmlVgpuInstanceSetEncoderCapacity	159
6.28.2.18	nvmlVgpuTypeGetClass	159
6.28.2.19	nvmlVgpuTypeGetDeviceID	160
6.28.2.20	nvmlVgpuTypeGetFramebufferSize	160
6.28.2.21	nvmlVgpuTypeGetFrameRateLimit	161
6.28.2.22	nvmlVgpuTypeGetLicense	161
6.28.2.23	nvmlVgpuTypeGetMaxInstances	162
6.28.2.24	nvmlVgpuTypeGetMaxInstancesPerVm	162
6.28.2.25	nvmlVgpuTypeGetName	162
6.28.2.26	nvmlVgpuTypeGetNumDisplayHeads	163
6.28.2.27	nvmlVgpuTypeGetResolution	163
6.29	GRID Virtualization Migration	164
6.29.1	Detailed Description	164

6.29.2	Enumeration Type Documentation	165
6.29.2.1	nvmlVgpuPgpuCompatibilityLimitCode_t	165
6.29.2.2	nvmlVgpuVmCompatibility_t	165
6.29.3	Function Documentation	165
6.29.3.1	nvmlDeviceGetPgpuMetadataString	165
6.29.3.2	nvmlDeviceGetVgpuMetadata	166
6.29.3.3	nvmlGetVgpuCompatibility	166
6.29.3.4	nvmlGetVgpuVersion	167
6.29.3.5	nvmlSetVgpuVersion	167
6.29.3.6	nvmlVgpuInstanceGetMetadata	168
6.30	GRID Virtualization Utilization and Accounting	169
6.30.1	Detailed Description	169
6.30.2	Function Documentation	169
6.30.2.1	nvmlDeviceGetVgpuProcessUtilization	169
6.30.2.2	nvmlDeviceGetVgpuUtilization	170
6.30.2.3	nvmlVgpuInstanceClearAccountingPids	171
6.30.2.4	nvmlVgpuInstanceGetAccountingMode	172
6.30.2.5	nvmlVgpuInstanceGetAccountingPids	172
6.30.2.6	nvmlVgpuInstanceGetAccountingStats	173
6.31	GPU Blacklist Queries	174
6.31.1	Detailed Description	174
6.31.2	Function Documentation	174
6.31.2.1	nvmlGetBlacklistDeviceCount	174
6.31.2.2	nvmlGetBlacklistDeviceInfoByIndex	174
6.32	Multi Instance GPU Management	176
6.32.1	Detailed Description	177
6.32.2	Define Documentation	177
6.32.2.1	NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE	177
6.32.2.2	NVML_DEVICE_MIG_DISABLE	177
6.32.2.3	NVML_DEVICE_MIG_ENABLE	177
6.32.2.4	NVML_GPU_INSTANCE_PROFILE_1_SLICE	177
6.32.3	Function Documentation	177
6.32.3.1	nvmlComputeInstanceDestroy	177
6.32.3.2	nvmlComputeInstanceGetInfo	178
6.32.3.3	nvmlDeviceCreateGpuInstance	178
6.32.3.4	nvmlDeviceGetComputeInstanceId	179
6.32.3.5	nvmlDeviceGetDeviceHandleFromMigDeviceHandle	179

6.32.3.6	nvmlDeviceGetGpuInstanceId	179
6.32.3.7	nvmlDeviceGetGpuInstanceId	180
6.32.3.8	nvmlDeviceGetGpuInstancePossiblePlacements	180
6.32.3.9	nvmlDeviceGetGpuInstanceProfileInfo	181
6.32.3.10	nvmlDeviceGetGpuInstanceRemainingCapacity	181
6.32.3.11	nvmlDeviceGetGpuInstances	182
6.32.3.12	nvmlDeviceGetMaxMigDeviceCount	182
6.32.3.13	nvmlDeviceGetMigDeviceHandleByIndex	182
6.32.3.14	nvmlDeviceGetMigMode	183
6.32.3.15	nvmlDeviceIsMigDeviceHandle	183
6.32.3.16	nvmlDeviceSetMigMode	184
6.32.3.17	nvmlGpuInstanceCreateComputeInstance	184
6.32.3.18	nvmlGpuInstanceDestroy	185
6.32.3.19	nvmlGpuInstanceGetComputeInstanceById	185
6.32.3.20	nvmlGpuInstanceGetComputeInstanceProfileInfo	186
6.32.3.21	nvmlGpuInstanceGetComputeInstanceRemainingCapacity	186
6.32.3.22	nvmlGpuInstanceGetComputeInstances	187
6.32.3.23	nvmlGpuInstanceGetInfo	187
6.33	NvmlClocksThrottleReasons	188
6.33.1	Define Documentation	188
6.33.1.1	nvmlClocksThrottleReasonAll	188
6.33.1.2	nvmlClocksThrottleReasonApplicationsClocksSetting	188
6.33.1.3	nvmlClocksThrottleReasonDisplayClockSetting	188
6.33.1.4	nvmlClocksThrottleReasonGpuIdle	189
6.33.1.5	nvmlClocksThrottleReasonHwPowerBrakeSlowdown	189
6.33.1.6	nvmlClocksThrottleReasonHwSlowdown	189
6.33.1.7	nvmlClocksThrottleReasonHwThermalSlowdown	189
6.33.1.8	nvmlClocksThrottleReasonNone	190
6.33.1.9	nvmlClocksThrottleReasonSwPowerCap	190
6.33.1.10	nvmlClocksThrottleReasonSwThermalSlowdown	190
6.33.1.11	nvmlClocksThrottleReasonSyncBoost	190
6.33.1.12	nvmlClocksThrottleReasonUserDefinedClocks	190
7	Data Structure Documentation	191
7.1	nvmlAccountingStats_t Struct Reference	191
7.1.1	Detailed Description	192
7.2	nvmlBAR1Memory_t Struct Reference	193

7.2.1 Detailed Description	193
7.3 nvmlBlacklistDeviceInfo_t Struct Reference	194
7.3.1 Detailed Description	194
7.4 nvmlBridgeChipHierarchy_t Struct Reference	195
7.4.1 Detailed Description	195
7.5 nvmlBridgeChipInfo_t Struct Reference	196
7.5.1 Detailed Description	196
7.6 nvmlEccErrorCounts_t Struct Reference	197
7.6.1 Detailed Description	197
7.7 nvmlEncoderSessionInfo_t Struct Reference	198
7.7.1 Detailed Description	198
7.8 nvmlEventData_t Struct Reference	199
7.8.1 Detailed Description	199
7.9 nvmlFBCSessionInfo_t Struct Reference	200
7.9.1 Detailed Description	200
7.10 nvmlFBCStats_t Struct Reference	201
7.10.1 Detailed Description	201
7.11 nvmlFieldValue_t Struct Reference	202
7.11.1 Detailed Description	202
7.12 nvmlGridLicensableFeature_t Struct Reference	203
7.12.1 Detailed Description	203
7.13 nvmlGridLicensableFeatures_t Struct Reference	204
7.13.1 Detailed Description	204
7.14 nvmlHwbcEntry_t Struct Reference	205
7.14.1 Detailed Description	205
7.15 nvmlLedState_t Struct Reference	206
7.15.1 Detailed Description	206
7.16 nvmlMemory_t Struct Reference	207
7.16.1 Detailed Description	207
7.17 nvmlNvLinkUtilizationControl_t Struct Reference	208
7.17.1 Detailed Description	208
7.18 nvmlPciInfo_t Struct Reference	209
7.18.1 Detailed Description	209
7.19 nvmlProcessInfo_t Struct Reference	210
7.19.1 Detailed Description	210
7.20 nvmlProcessUtilizationSample_t Struct Reference	211
7.20.1 Detailed Description	211

7.21	nvmlPSUInfo_t Struct Reference	212
7.21.1	Detailed Description	212
7.22	nvmlRowMapperHistogramValues_t Struct Reference	213
7.22.1	Detailed Description	213
7.23	nvmlSample_t Struct Reference	214
7.23.1	Detailed Description	214
7.24	nvmlUnitFanInfo_t Struct Reference	215
7.24.1	Detailed Description	215
7.25	nvmlUnitFanSpeeds_t Struct Reference	216
7.25.1	Detailed Description	216
7.26	nvmlUnitInfo_t Struct Reference	217
7.26.1	Detailed Description	217
7.27	nvmlUtilization_t Struct Reference	218
7.27.1	Detailed Description	218
7.28	nvmlValue_t Union Reference	219
7.28.1	Detailed Description	219
7.29	nvmlVgpuInstanceUtilizationSample_t Struct Reference	220
7.29.1	Detailed Description	220
7.30	nvmlVgpuMetadata_t Struct Reference	221
7.30.1	Detailed Description	221
7.31	nvmlVgpuPgpuCompatibility_t Struct Reference	222
7.31.1	Detailed Description	222
7.32	nvmlVgpuPgpuMetadata_t Struct Reference	223
7.32.1	Detailed Description	223
7.33	nvmlVgpuProcessUtilizationSample_t Struct Reference	224
7.33.1	Detailed Description	224
7.34	nvmlVgpuVersion_t Struct Reference	225
7.34.1	Detailed Description	225
7.35	nvmlViolationTime_t Struct Reference	226
7.35.1	Detailed Description	226

Chapter 1

Known issues in the current version of NVML library

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

- On Linux GPU Reset can't be triggered when there is pending GPU Operation Mode (GOM) change
- On Linux GPU Reset may not successfully change pending ECC mode. A full reboot may be required to enable the mode change.
- [Accounting Statistics](#) supports only one process per GPU at a time (CUDA proxy server counts as one process).
- [nvmlAccountingStats_t::time](#) reports time and utilization values starting from cuInit till process termination. Next driver versions might change this behavior slightly and account process only from cuCtxCreate till cuCtxDestroy.
- On GPUs from Fermi family current P0 clocks (reported by [nvmlDeviceGetClockInfo](#)) can differ from max clocks by few MHz.

Chapter 2

Change log of NVML library

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

2.1 Changes between nvidia-smi v445 Update and v450 ===

- Updated [nvmlDeviceGetFanSpeed](#) and [nvmlDeviceGetFanSpeed_v2](#) for allowing fan speeds greater than 100% to be reported.
- Added [nvmlDeviceGetCpuAffinityWithinScope](#) to determine the closest processor(s) within a NUMA node or socket.
- Added [nvmlDeviceGetMemoryAffinity](#) to determine the closest NUMA node(s) within a NUMA node or socket.

2.2 Changes between nvidia-smi v418 Update and v445 ===

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

2.3 Changes between nvidia-smi v361 Update and v418

- Support for Volta and Turing architectures, bug fixes, performance improvements, and new features

2.4 Changes between nvidia-smi v349 Update and v361

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

2.5 Changes between NVML v346 Update and v349

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

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

2.6 Changes between NVML v340 Update and v346

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

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

2.7 Changes between NVML v331 Update and v340

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

- Added `nvmlDeviceGetSamples` to get recent power, utilization and clock samples for the GPU.
- Added `nvmlDeviceGetTemperatureThreshold` to retrieve temperature threshold information.
- Added `nvmlDeviceGetBrand` to retrieve brand information (e.g. Tesla, Quadro, etc.)
- Added support for K40d and K80
- Added `nvmlDeviceGetTopology` internal API to retrieve path info between PCI devices (remove this for DITA)
- Added `nvmlDeviceGetViolationStatus` to get the duration of time during which the device was throttled (lower than requested clocks) due to thermal or power constraints.
- Added `nvmlDeviceGetEncoderUtilization` and `nvmlDeviceGetDecoderUtilization` APIs
- Added `nvmlDeviceGetCpuAffinity` to determine the closest processor(s) affinity to a specific GPU
- Added `nvmlDeviceSetCpuAffinity` to bind a specific GPU to the closest processor
- Added `nvmlDeviceClearCpuAffinity` to unbind a specific GPU
- Added `nvmlDeviceGetBoardId` to get a unique boardId for the running system
- Added `nvmlDeviceGetMultiGpuBoard` to get whether the device is on a multiGPU board
- Added `nvmlDeviceGetAutoBoostedClocksEnabled` and `nvmlDeviceSetAutoBoostedClocksEnabled` for querying and setting the state of auto boosted clocks on supporting hardware.
- Added `nvmlDeviceSetDefaultAutoBoostedClocksEnabled` for setting the default state of auto boosted clocks on supporting hardware.

2.8 Changes between NVML v5.319 Update and v331

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

- Added `nvmlDeviceGetMinorNumber` to get the minor number for the device.
- Added `nvmlDeviceGetBAR1MemoryInfo` to get BAR1 total, available and used memory size.
- Added `nvmlDeviceGetBridgeChipInfo` to get the information related to bridge chip firmware.
- Added enforced power limit query API `nvmlDeviceGetEnforcedPowerLimit`
- Updated `nvmlEventSetWait_v2` to return xid event data in case of xid error event.
- Added support for K8

2.9 Changes between NVML v5.319 RC and v5.319 Update

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

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

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

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

- IMPORTANT: Added `_v2` versions of [nvmlDeviceGetHandleByIndex_v2](#) and [nvmlDeviceGetCount_v2](#) that also count devices not accessible by current user
 - IMPORTANT: [nvmlDeviceGetHandleByIndex_v2](#) (default) can also return `NVML_ERROR_NO_PERMISSION`
- Added [nvmlInit_v2](#) and [nvmlDeviceGetHandleByIndex_v2](#) that is safer and thus recommended function for initializing the library
 - [nvmlInit_v2](#) lazily initializes only requested devices (queried with [nvmlDeviceGetHandle*](#))
 - `nvml.h` defines [nvmlInit_v2](#) and [nvmlDeviceGetHandleByIndex_v2](#) as default functions
- Added [nvmlDeviceGetIndex](#)
- Added [NVML_ERROR_GPU_IS_LOST](#) to report GPUs that have fallen off the bus.
 - Note: All NVML device APIs can return this error code, as a GPU can fall off the bus at any time.
- Added new class of APIs for gathering process statistics ([Accounting Statistics](#))
- Application Clocks are no longer supported on GPU's from Quadro product line
- Added APIs to support dynamic page retirement. See [nvmlDeviceGetRetiredPages](#) and [nvmlDeviceGetRetiredPagesPendingStatus](#)
- Renamed `nvmlClocksThrottleReasonUserDefinedClocks` to `nvmlClocksThrottleReasonApplicationsClocksSetting`. Old name is deprecated and can be removed in one of the next major releases.
- Added [nvmlDeviceGetDisplayActive](#) and updated documentation to clarify how it differs from [nvmlDeviceGetDisplayMode](#)

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

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

- Added [nvmlDeviceGetGpuOperationMode](#) and [nvmlDeviceSetGpuOperationMode](#)

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

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

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

2.13 Changes between NVML v2.285 and v3.295

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

- deprecated [nvmlDeviceGetHandleBySerial](#) in favor of newly added [nvmlDeviceGetHandleByUUID](#)
- Marked the input parameters of [nvmlDeviceGetHandleBySerial](#), [nvmlDeviceGetHandleByUUID](#) and [nvmlDeviceGetHandleByPciBusId_v2](#) as const
- Added [nvmlDeviceOnSameBoard](#)
- Added [Constants](#) defines
- Added [nvmlDeviceGetMaxPcieLinkGeneration](#), [nvmlDeviceGetMaxPcieLinkWidth](#), [nvmlDeviceGetCurrPcieLinkGeneration](#), [nvmlDeviceGetCurrPcieLinkWidth](#)
- Format change of [nvmlDeviceGetUUID](#) output to match the UUID standard. This function will return a different value.
- [nvmlDeviceGetDetailedEccErrors](#) will report zero for unsupported ECC error counters when a subset of ECC error counters are supported

2.14 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_v2](#)
- 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_v2](#) is less restrictive. You can pass 0:2:0.0 or 0000:02:00 and other variations
- Added API for events waiting for GPU events (Linux only) see docs of [Event Handling Methods](#)
- Added API [nvmlDeviceGetComputeRunningProcesses](#) and [nvmlSystemGetProcessName](#) functions for looking up currently running compute applications
- Deprecated [nvmlDeviceGetPowerState](#) in favor of [nvmlDeviceGetPerformanceState](#).

Chapter 3

Deprecated List

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

Global [NVML_DOUBLE_BIT_ECC](#) Mapped to [NVML_MEMORY_ERROR_TYPE_UNCORRECTED](#)

Global [NVML_SINGLE_BIT_ECC](#) Mapped to [NVML_MEMORY_ERROR_TYPE_CORRECTED](#)

Global [nvmlEccBitType_t](#) See [nvmlMemoryErrorType_t](#) for a more flexible type

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

Global [nvmlDeviceGetHandleBySerial](#) Since more than one GPU can exist on a single board this function is deprecated in favor of [nvmlDeviceGetHandleByUUID](#). For dual GPU boards this function will return [NVML_ERROR_INVALID_ARGUMENT](#).

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

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

Device Structs	15
Device Enums	19
Field Value Enums	32
Unit Structs	42
Accounting Statistics	45
Encoder Structs	49
Frame Buffer Capture Structures	50
definitions related to the drain state	51
Initialization and Cleanup	52
Error reporting	54
Constants	55
System Queries	57
Unit Queries	60
Device Queries	64
CPU and Memory Affinity	118
Unit Commands	122
Device Commands	123
NvLink Methods	132
Event Handling Methods	138
Event Types	43
Drain states	142
Field Value Queries	145
GRID Virtualization Enums, Constants and Structs	146
GRID Virtualization Enums	28
GRID Virtualization Constants	30
GRID Virtualization Structs	31
GRID Virtualization APIs	147
GRID vGPU Management	150
GRID Virtualization Migration	164
GRID Virtualization Utilization and Accounting	169
GPU Blacklist Queries	174
Multi Instance GPU Management	176
NvmlClocksThrottleReasons	188

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

nvmlAccountingStats_t	191
nvmlBAR1Memory_t	193
nvmlBlacklistDeviceInfo_t	194
nvmlBridgeChipHierarchy_t	195
nvmlBridgeChipInfo_t	196
nvmlEccErrorCounts_t	197
nvmlEncoderSessionInfo_t	198
nvmlEventData_t	199
nvmlFBCSessionInfo_t	200
nvmlFBCStats_t	201
nvmlFieldValue_t	202
nvmlGridLicensableFeature_t	203
nvmlGridLicensableFeatures_t	204
nvmlHwbcEntry_t	205
nvmlLedState_t	206
nvmlMemory_t	207
nvmlNvLinkUtilizationControl_t	208
nvmlPciInfo_t	209
nvmlProcessInfo_t	210
nvmlProcessUtilizationSample_t	211
nvmlPSUInfo_t	212
nvmlRowRemapperHistogramValues_t	213
nvmlSample_t	214
nvmlUnitFanInfo_t	215
nvmlUnitFanSpeeds_t	216
nvmlUnitInfo_t	217
nvmlUtilization_t	218
nvmlValue_t	219
nvmlVgpuInstanceUtilizationSample_t	220
nvmlVgpuMetadata_t	221
nvmlVgpuPgpuCompatibility_t	222
nvmlVgpuPgpuMetadata_t	223
nvmlVgpuProcessUtilizationSample_t	224

nvmlVgpuVersion_t	225
nvmlViolationTime_t	226

Chapter 6

Module Documentation

6.1 Device Structs

Data Structures

- struct [nvmlPciInfo_t](#)
- struct [nvmlEccErrorCounts_t](#)
- struct [nvmlUtilization_t](#)
- struct [nvmlMemory_t](#)
- struct [nvmlBAR1Memory_t](#)
- struct [nvmlProcessInfo_t](#)
- struct [nvmlRowRemapperHistogramValues_t](#)
- struct [nvmlNvLinkUtilizationControl_t](#)
- struct [nvmlBridgeChipInfo_t](#)
- struct [nvmlBridgeChipHierarchy_t](#)
- union [nvmlValue_t](#)
- struct [nvmlSample_t](#)
- struct [nvmlViolationTime_t](#)

Defines

- #define [NVML_VALUE_NOT_AVAILABLE](#) (-1)
- #define [NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE](#) 32
- #define [NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE](#) 16
- #define [NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT](#) "%04X:%02X:%02X.0"
- #define [NVML_DEVICE_PCI_BUS_ID_FMT](#) "%08X:%02X:%02X.0"
- #define [NVML_DEVICE_PCI_BUS_ID_FMT_ARGS](#)(pciInfo)
- #define [NVML_NVLINK_MAX_LINKS](#) 12
- #define [NVML_MAX_PHYSICAL_BRIDGE](#) (128)

Enumerations

- enum [nvmlBridgeChipType_t](#)
- enum [nvmlNvLinkUtilizationCountUnits_t](#)
- enum [nvmlNvLinkUtilizationCountPktTypes_t](#)

- enum [nvmlNvLinkCapability_t](#)
- enum [nvmlNvLinkErrorCounter_t](#)
- enum [nvmlGpuTopologyLevel_t](#)
- enum [nvmlSamplingType_t](#) {
[NVML_TOTAL_POWER_SAMPLES](#) = 0,
[NVML_GPU_UTILIZATION_SAMPLES](#) = 1,
[NVML_MEMORY_UTILIZATION_SAMPLES](#) = 2,
[NVML_ENC_UTILIZATION_SAMPLES](#) = 3,
[NVML_DEC_UTILIZATION_SAMPLES](#) = 4,
[NVML_PROCESSOR_CLK_SAMPLES](#) = 5,
[NVML_MEMORY_CLK_SAMPLES](#) = 6 }
- enum [nvmlPcieUtilCounter_t](#)
- enum [nvmlValueType_t](#)
- enum [nvmlPerfPolicyType_t](#) {
[NVML_PERF_POLICY_POWER](#) = 0,
[NVML_PERF_POLICY_THERMAL](#) = 1,
[NVML_PERF_POLICY_SYNC_BOOST](#) = 2,
[NVML_PERF_POLICY_BOARD_LIMIT](#) = 3,
[NVML_PERF_POLICY_LOW_UTILIZATION](#) = 4,
[NVML_PERF_POLICY_RELIABILITY](#) = 5,
[NVML_PERF_POLICY_TOTAL_APP_CLOCKS](#) = 10,
[NVML_PERF_POLICY_TOTAL_BASE_CLOCKS](#) = 11 }

6.1.1 Define Documentation

6.1.1.1 `#define NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE 32`

Buffer size guaranteed to be large enough for pci bus id

6.1.1.2 `#define NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE 16`

Buffer size guaranteed to be large enough for pci bus id for busIdLegacy

6.1.1.3 `#define NVML_DEVICE_PCI_BUS_ID_FMT "%08X:%02X:%02X.0"`

PCI format string for busId

6.1.1.4 `#define NVML_DEVICE_PCI_BUS_ID_FMT_ARGS(pciInfo)`

Value:

```
(pciInfo)->domain, \
                                (pciInfo)->bus, \
                                (pciInfo)->device
```

Utility macro for filling the pci bus id format from a [nvmlPciInfo_t](#)

6.1.1.5 #define NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT "%04X:%02X:%02X.0"

PCI format string for busIdLegacy

6.1.1.6 #define NVML_MAX_PHYSICAL_BRIDGE (128)

Maximum limit on Physical Bridges per Board

6.1.1.7 #define NVML_NVLINK_MAX_LINKS 12

Maximum number of NvLink links supported

6.1.1.8 #define NVML_VALUE_NOT_AVAILABLE (-1)

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

6.1.2 Enumeration Type Documentation**6.1.2.1 enum nvmlBridgeChipType_t**

Enum to represent type of bridge chip

6.1.2.2 enum nvmlGpuTopologyLevel_t

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

6.1.2.3 enum nvmlNvLinkCapability_t

Enum to represent NvLink queryable capabilities

6.1.2.4 enum nvmlNvLinkErrorCounter_t

Enum to represent NvLink queryable error counters

6.1.2.5 enum nvmlNvLinkUtilizationCountPktTypes_t

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

6.1.2.6 enum nvmlNvLinkUtilizationCountUnits_t

Enum to represent the NvLink utilization counter packet units

6.1.2.7 enum nvmlPcieUtilCounter_t

Represents the queryable PCIe utilization counters

6.1.2.8 enum nvmlPerfPolicyType_t

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

Enumerator:

NVML_PERF_POLICY_POWER How long did power violations cause the GPU to be below application clocks.

NVML_PERF_POLICY_THERMAL How long did thermal violations cause the GPU to be below application clocks.

NVML_PERF_POLICY_SYNC_BOOST How long did sync boost cause the GPU to be below application clocks.

NVML_PERF_POLICY_BOARD_LIMIT How long did the board limit cause the GPU to be below application clocks.

NVML_PERF_POLICY_LOW_UTILIZATION How long did low utilization cause the GPU to be below application clocks.

NVML_PERF_POLICY_RELIABILITY How long did the board reliability limit cause the GPU to be below application clocks.

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

NVML_PERF_POLICY_TOTAL_BASE_CLOCKS Total time the GPU was held below base clocks.

6.1.2.9 enum nvmlSamplingType_t

Represents Type of Sampling Event

Enumerator:

NVML_TOTAL_POWER_SAMPLES To represent total power drawn by GPU.

NVML_GPU_UTILIZATION_SAMPLES To represent percent of time during which one or more kernels was executing on the GPU.

NVML_MEMORY_UTILIZATION_SAMPLES To represent percent of time during which global (device) memory was being read or written.

NVML_ENC_UTILIZATION_SAMPLES To represent percent of time during which NVENC remains busy.

NVML_DEC_UTILIZATION_SAMPLES To represent percent of time during which NVDEC remains busy.

NVML_PROCESSOR_CLK_SAMPLES To represent processor clock samples.

NVML_MEMORY_CLK_SAMPLES To represent memory clock samples.

6.1.2.10 enum nvmlValueType_t

Represents the type for sample value returned

6.2 Device Enums

Defines

- `#define nvmlFlagDefault 0x00`
Generic flag used to specify the default behavior of some functions. See description of particular functions for details.
- `#define nvmlFlagForce 0x01`
Generic flag used to force some behavior. See description of particular functions for details.
- `#define nvmlEccBitType_t nvmlMemoryErrorType_t`
- `#define NVML_SINGLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_CORRECTED`
- `#define NVML_DOUBLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_UNCORRECTED`

Enumerations

- `enum nvmlEnableState_t {`
 `NVML_FEATURE_DISABLED = 0,`
 `NVML_FEATURE_ENABLED = 1 }`
- `enum nvmlBrandType_t`
- `enum nvmlTemperatureThresholds_t`
- `enum nvmlTemperatureSensors_t { NVML_TEMPERATURE_GPU = 0 }`
- `enum nvmlComputeMode_t {`
 `NVML_COMPUTEMODE_DEFAULT = 0,`
 `NVML_COMPUTEMODE_EXCLUSIVE_THREAD = 1,`
 `NVML_COMPUTEMODE_PROHIBITED = 2,`
 `NVML_COMPUTEMODE_EXCLUSIVE_PROCESS = 3 }`
- `enum nvmlMemoryErrorType_t {`
 `NVML_MEMORY_ERROR_TYPE_CORRECTED = 0,`
 `NVML_MEMORY_ERROR_TYPE_UNCORRECTED = 1,`
 `NVML_MEMORY_ERROR_TYPE_COUNT }`
- `enum nvmlEccCounterType_t {`
 `NVML_VOLATILE_ECC = 0,`
 `NVML_AGGREGATE_ECC = 1,`
 `NVML_ECC_COUNTER_TYPE_COUNT }`
- `enum nvmlClockType_t {`
 `NVML_CLOCK_GRAPHICS = 0,`
 `NVML_CLOCK_SM = 1,`
 `NVML_CLOCK_MEM = 2,`
 `NVML_CLOCK_VIDEO = 3,`
 `NVML_CLOCK_COUNT }`
- `enum nvmlClockId_t {`
 `NVML_CLOCK_ID_CURRENT = 0,`
 `NVML_CLOCK_ID_APP_CLOCK_TARGET = 1,`
 `NVML_CLOCK_ID_APP_CLOCK_DEFAULT = 2,`
 `NVML_CLOCK_ID_CUSTOMER_BOOST_MAX = 3,`
 `NVML_CLOCK_ID_COUNT }`

- enum `nvmlDriverModel_t` {
 `NVML_DRIVER_WDDM` = 0,
 `NVML_DRIVER_WDM` = 1 }
- enum `nvmlPstates_t` {
 `NVML_PSTATE_0` = 0,
 `NVML_PSTATE_1` = 1,
 `NVML_PSTATE_2` = 2,
 `NVML_PSTATE_3` = 3,
 `NVML_PSTATE_4` = 4,
 `NVML_PSTATE_5` = 5,
 `NVML_PSTATE_6` = 6,
 `NVML_PSTATE_7` = 7,
 `NVML_PSTATE_8` = 8,
 `NVML_PSTATE_9` = 9,
 `NVML_PSTATE_10` = 10,
 `NVML_PSTATE_11` = 11,
 `NVML_PSTATE_12` = 12,
 `NVML_PSTATE_13` = 13,
 `NVML_PSTATE_14` = 14,
 `NVML_PSTATE_15` = 15,
 `NVML_PSTATE_UNKNOWN` = 32 }
- enum `nvmlGpuOperationMode_t` {
 `NVML_GOM_ALL_ON` = 0,
 `NVML_GOM_COMPUTE` = 1,
 `NVML_GOM_LOW_DP` = 2 }
- enum `nvmlInforomObject_t` {
 `NVML_INFOROM_OEM` = 0,
 `NVML_INFOROM_ECC` = 1,
 `NVML_INFOROM_POWER` = 2,
 `NVML_INFOROM_COUNT` }
- enum `nvmlReturn_t` {
 `NVML_SUCCESS` = 0,
 `NVML_ERROR_UNINITIALIZED` = 1,
 `NVML_ERROR_INVALID_ARGUMENT` = 2,
 `NVML_ERROR_NOT_SUPPORTED` = 3,
 `NVML_ERROR_NO_PERMISSION` = 4,
 `NVML_ERROR_ALREADY_INITIALIZED` = 5,
 `NVML_ERROR_NOT_FOUND` = 6,
 `NVML_ERROR_INSUFFICIENT_SIZE` = 7,
 `NVML_ERROR_INSUFFICIENT_POWER` = 8,
 `NVML_ERROR_DRIVER_NOT_LOADED` = 9,
 `NVML_ERROR_TIMEOUT` = 10,

```

NVML_ERROR_IRQ_ISSUE = 11,
NVML_ERROR_LIBRARY_NOT_FOUND = 12,
NVML_ERROR_FUNCTION_NOT_FOUND = 13,
NVML_ERROR_CORRUPTED_INFOROM = 14,
NVML_ERROR_GPU_IS_LOST = 15,
NVML_ERROR_RESET_REQUIRED = 16,
NVML_ERROR_OPERATING_SYSTEM = 17,
NVML_ERROR_LIB_RM_VERSION_MISMATCH = 18,
NVML_ERROR_IN_USE = 19,
NVML_ERROR_MEMORY = 20,
NVML_ERROR_NO_DATA = 21,
NVML_ERROR_VGPU_ECC_NOT_SUPPORTED = 22,
NVML_ERROR_INSUFFICIENT_RESOURCES = 23,
NVML_ERROR_UNKNOWN = 999 }
• enum nvmlMemoryLocation_t {
    NVML_MEMORY_LOCATION_L1_CACHE = 0,
    NVML_MEMORY_LOCATION_L2_CACHE = 1,
    NVML_MEMORY_LOCATION_DRAM = 2,
    NVML_MEMORY_LOCATION_DEVICE_MEMORY = 2,
    NVML_MEMORY_LOCATION_REGISTER_FILE = 3,
    NVML_MEMORY_LOCATION_TEXTURE_MEMORY = 4,
    NVML_MEMORY_LOCATION_TEXTURE_SHM = 5,
    NVML_MEMORY_LOCATION_CBU = 6,
    NVML_MEMORY_LOCATION_SRAM = 7,
    NVML_MEMORY_LOCATION_COUNT }
• enum nvmlPageRetirementCause_t {
    NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS = 0,
    NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR = 1 }
• enum nvmlRestrictedAPI_t {
    NVML_RESTRICTED_API_SET_APPLICATION_CLOCKS = 0,
    NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS = 1 }

```

6.2.1 Define Documentation

6.2.1.1 #define NVML_DOUBLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_UNCORRECTED

Double bit ECC errors

Deprecated

Mapped to `NVML_MEMORY_ERROR_TYPE_UNCORRECTED`

6.2.1.2 `#define NVML_SINGLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_CORRECTED`

Single bit ECC errors

Deprecated

Mapped to [NVML_MEMORY_ERROR_TYPE_CORRECTED](#)

6.2.1.3 `#define nvmlEccBitType_t nvmlMemoryErrorType_t`

ECC bit types.

Deprecated

See [nvmlMemoryErrorType_t](#) for a more flexible type

6.2.2 Enumeration Type Documentation

6.2.2.1 `enum nvmlBrandType_t`

* The Brand of the GPU

6.2.2.2 `enum nvmlClockId_t`

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

Enumerator:

NVML_CLOCK_ID_CURRENT Current actual clock value.

NVML_CLOCK_ID_APP_CLOCK_TARGET Target application clock.

NVML_CLOCK_ID_APP_CLOCK_DEFAULT Default application clock target.

NVML_CLOCK_ID_CUSTOMER_BOOST_MAX OEM-defined maximum clock rate.

NVML_CLOCK_ID_COUNT Count of Clock Ids.

6.2.2.3 `enum nvmlClockType_t`

Clock types.

All speeds are in Mhz.

Enumerator:

NVML_CLOCK_GRAPHICS Graphics clock domain.

NVML_CLOCK_SM SM clock domain.

NVML_CLOCK_MEM Memory clock domain.

NVML_CLOCK_VIDEO Video encoder/decoder clock domain.

NVML_CLOCK_COUNT Count of clock types.

6.2.2.4 enum nvmlComputeMode_t

Compute mode.

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

Enumerator:

NVML_COMPUTEMODE_DEFAULT Default compute mode – multiple contexts per device.

NVML_COMPUTEMODE_EXCLUSIVE_THREAD Support Removed.

NVML_COMPUTEMODE_PROHIBITED Compute-prohibited mode – no contexts per device.

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

6.2.2.5 enum nvmlDriverModel_t

Driver models.

Windows only.

Enumerator:

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

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

6.2.2.6 enum nvmlEccCounterType_t

ECC counter types.

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

Enumerator:

NVML_VOLATILE_ECC Volatile counts are reset each time the driver loads.

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

NVML_ECC_COUNTER_TYPE_COUNT Count of memory counter types.

6.2.2.7 enum nvmlEnableState_t

Generic enable/disable enum.

Enumerator:

NVML_FEATURE_DISABLED Feature disabled.

NVML_FEATURE_ENABLED Feature enabled.

6.2.2.8 enum nvmlGpuOperationMode_t

GPU Operation Mode

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

Each GOM is designed to meet specific user needs.

Enumerator:

NVML_GOM_ALL_ON Everything is enabled and running at full speed.

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

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

6.2.2.9 enum nvmlInforomObject_t

Available infoROM objects.

Enumerator:

NVML_INFOTOM_OEM An object defined by OEM.

NVML_INFOTOM_ECC The ECC object determining the level of ECC support.

NVML_INFOTOM_POWER The power management object.

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

6.2.2.10 enum nvmlMemoryErrorType_t

Memory error types

Enumerator:

NVML_MEMORY_ERROR_TYPE_CORRECTED A memory error that was corrected

For ECC errors, these are single bit errors For Texture memory, these are errors fixed by resend

NVML_MEMORY_ERROR_TYPE_UNCORRECTED A memory error that was not corrected

For ECC errors, these are double bit errors For Texture memory, these are errors where the resend fails

NVML_MEMORY_ERROR_TYPE_COUNT Count of memory error types.

6.2.2.11 enum nvmlMemoryLocation_t

See [nvmlDeviceGetMemoryErrorCounter](#)

Enumerator:

NVML_MEMORY_LOCATION_L1_CACHE GPU L1 Cache.

NVML_MEMORY_LOCATION_L2_CACHE GPU L2 Cache.

NVML_MEMORY_LOCATION_DRAM Turing+ DRAM.

NVML_MEMORY_LOCATION_DEVICE_MEMORY GPU Device Memory.

NVML_MEMORY_LOCATION_REGISTER_FILE GPU Register File.

NVML_MEMORY_LOCATION_TEXTURE_MEMORY GPU Texture Memory.

NVML_MEMORY_LOCATION_TEXTURE_SHM Shared memory.

NVML_MEMORY_LOCATION_CBU CBU.

NVML_MEMORY_LOCATION_SRAM Turing+ SRAM.

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

6.2.2.12 enum nvmlPageRetirementCause_t

Causes for page retirement

Enumerator:

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

NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR Page was retired due to double bit ECC error.

6.2.2.13 enum nvmlPstates_t

Allowed PStates.

Enumerator:

NVML_PSTATE_0 Performance state 0 – Maximum Performance.

NVML_PSTATE_1 Performance state 1.

NVML_PSTATE_2 Performance state 2.

NVML_PSTATE_3 Performance state 3.

NVML_PSTATE_4 Performance state 4.

NVML_PSTATE_5 Performance state 5.

NVML_PSTATE_6 Performance state 6.

NVML_PSTATE_7 Performance state 7.

NVML_PSTATE_8 Performance state 8.

NVML_PSTATE_9 Performance state 9.

NVML_PSTATE_10 Performance state 10.

NVML_PSTATE_11 Performance state 11.

NVML_PSTATE_12 Performance state 12.

NVML_PSTATE_13 Performance state 13.

NVML_PSTATE_14 Performance state 14.

NVML_PSTATE_15 Performance state 15 – Minimum Performance.

NVML_PSTATE_UNKNOWN Unknown performance state.

6.2.2.14 enum nvmlRestrictedAPI_t

API types that allow changes to default permission restrictions

Enumerator:

NVML_RESTRICTED_API_SET_APPLICATION_CLOCKS APIs that change application clocks, see `nvmlDeviceSetApplicationsClocks` < and see `nvmlDeviceResetApplicationsClocks`.

NVML_RESTRICTED_API_SET_AUTO_BOOSTED_CLOCKS APIs that enable/disable Auto Boosted clocks < see `nvmlDeviceSetAutoBoostedClocksEnabled`.

6.2.2.15 enum nvmlReturn_t

Return values for NVML API calls.

Enumerator:

NVML_SUCCESS The operation was successful.

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

NVML_ERROR_INVALID_ARGUMENT A supplied argument is invalid.

NVML_ERROR_NOT_SUPPORTED The requested operation is not available on target device.

NVML_ERROR_NO_PERMISSION The current user does not have permission for operation.

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

NVML_ERROR_NOT_FOUND A query to find an object was unsuccessful.

NVML_ERROR_INSUFFICIENT_SIZE An input argument is not large enough.

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

NVML_ERROR_DRIVER_NOT_LOADED NVIDIA driver is not loaded.

NVML_ERROR_TIMEOUT User provided timeout passed.

NVML_ERROR_IRQ_ISSUE NVIDIA Kernel detected an interrupt issue with a GPU.

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

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

NVML_ERROR_CORRUPTED_INFOROM infoROM is corrupted

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

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

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

NVML_ERROR_LIB_RM_VERSION_MISMATCH RM detects a driver/library version mismatch.

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

NVML_ERROR_MEMORY Insufficient memory.

NVML_ERROR_NO_DATA No data.

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

NVML_ERROR_INSUFFICIENT_RESOURCES Ran out of critical resources, other than memory.

NVML_ERROR_UNKNOWN An internal driver error occurred.

6.2.2.16 enum nvmlTemperatureSensors_t

Temperature sensors.

Enumerator:

NVML_TEMPERATURE_GPU Temperature sensor for the GPU die.

6.2.2.17 enum nvmlTemperatureThresholds_t

Temperature thresholds.

6.3 GRID Virtualization Enums

Enumerations

- enum `nvmlGpuVirtualizationMode_t` {
`NVML_GPU_VIRTUALIZATION_MODE_NONE` = 0,
`NVML_GPU_VIRTUALIZATION_MODE_PASSTHROUGH` = 1,
`NVML_GPU_VIRTUALIZATION_MODE_VGPU` = 2,
`NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU` = 3,
`NVML_GPU_VIRTUALIZATION_MODE_HOST_VSGA` = 4 }
- enum `nvmlHostVgpuMode_t` {
`NVML_HOST_VGPU_MODE_NON_SRIOV` = 0,
`NVML_HOST_VGPU_MODE_SRIOV` = 1 }
- enum `nvmlVgpuVmIdType_t` {
`NVML_VGPU_VM_ID_DOMAIN_ID` = 0,
`NVML_VGPU_VM_ID_UUID` = 1 }
- enum `nvmlVgpuGuestInfoState_t` {
`NVML_VGPU_INSTANCE_GUEST_INFO_STATE_UNINITIALIZED` = 0,
`NVML_VGPU_INSTANCE_GUEST_INFO_STATE_INITIALIZED` = 1 }
- enum `nvmlGridLicenseFeatureCode_t` {
`NVML_GRID_LICENSE_FEATURE_CODE_VGPU` = 1,
`NVML_GRID_LICENSE_FEATURE_CODE_VWORKSTATION` = 2 }

6.3.1 Enumeration Type Documentation

6.3.1.1 enum `nvmlGpuVirtualizationMode_t`

GPU virtualization mode types.

Enumerator:

- `NVML_GPU_VIRTUALIZATION_MODE_NONE`** Represents Bare Metal GPU.
- `NVML_GPU_VIRTUALIZATION_MODE_PASSTHROUGH`** Device is associated with GPU-Passthrough.
- `NVML_GPU_VIRTUALIZATION_MODE_VGPU`** Device is associated with vGPU inside virtual machine.
- `NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU`** Device is associated with VGX hypervisor in vGPU mode.
- `NVML_GPU_VIRTUALIZATION_MODE_HOST_VSGA`** Device is associated with VGX hypervisor in vSGA mode.

6.3.1.2 enum `nvmlGridLicenseFeatureCode_t`

GRID license feature code

Enumerator:

- `NVML_GRID_LICENSE_FEATURE_CODE_VGPU`** Virtual GPU.
- `NVML_GRID_LICENSE_FEATURE_CODE_VWORKSTATION`** Virtual Workstation.

6.3.1.3 enum nvmlHostVgpuMode_t

Host vGPU modes

Enumerator:

NVML_HOST_VGPU_MODE_NON_SRIOV Non SR-IOV mode.

NVML_HOST_VGPU_MODE_SRIOV SR-IOV mode.

6.3.1.4 enum nvmlVgpuGuestInfoState_t

vGPU GUEST info state.

Enumerator:

NVML_VGPU_INSTANCE_GUEST_INFO_STATE_UNINITIALIZED Guest-dependent fields uninitialized.

NVML_VGPU_INSTANCE_GUEST_INFO_STATE_INITIALIZED Guest-dependent fields initialized.

6.3.1.5 enum nvmlVgpuVmIdType_t

Types of VM identifiers

Enumerator:

NVML_VGPU_VM_ID_DOMAIN_ID VM ID represents DOMAIN ID.

NVML_VGPU_VM_ID_UUID VM ID represents UUID.

6.4 GRID Virtualization Constants

Defines

- `#define NVML_GRID_LICENSE_BUFFER_SIZE 128`
- `#define NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION 0:0`
- `#define NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION 0:0`

6.4.1 Define Documentation

6.4.1.1 `#define NVML_GRID_LICENSE_BUFFER_SIZE 128`

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

6.4.1.2 `#define NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION 0:0`

Macros for pGPU's virtualization capabilities bitfield.

6.4.1.3 `#define NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION 0:0`

Macros for vGPU instance's virtualization capabilities bitfield.

6.5 GRID Virtualization Structs

Data Structures

- struct [nvmlVgpuInstanceUtilizationSample_t](#)
- struct [nvmlVgpuProcessUtilizationSample_t](#)
- struct [nvmlProcessUtilizationSample_t](#)
- struct [nvmlGridLicensableFeature_t](#)
- struct [nvmlGridLicensableFeatures_t](#)

Defines

- [#define NVML_DEVICE_ARCH_KEPLER 2](#)

6.5.1 Define Documentation

6.5.1.1 [#define NVML_DEVICE_ARCH_KEPLER 2](#)

Simplified chip architecture

6.6 Field Value Enums

Data Structures

- struct [nvmlFieldValue_t](#)

Defines

- #define [NVML_FI_DEV_ECC_CURRENT](#) 1
Current ECC mode. 1=Active. 0=Inactive.
- #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_NVLINK_REPLAY_ERROR_COUNT_L3` 49
NVLink Replay Error Counter for Lane 3.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L4` 50
NVLink Replay Error Counter for Lane 4.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L5` 51
NVLink Replay Error Counter for Lane 5.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_TOTAL` 52
NVLink Replay Error Counter total for all Lanes.
- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L0` 53
NVLink Recovery Error Counter for Lane 0.
- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L1` 54
NVLink Recovery Error Counter for Lane 1.
- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L2` 55
NVLink Recovery Error Counter for Lane 2.
- `#define NVML_FI_DEV_NVLINK_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_APP_CLOCKS` 80
Perf Policy Counter for Total App Clock Policy.
- `#define NVML_FI_DEV_PERF_POLICY_TOTAL_BASE_CLOCKS` 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_TOTAL_ENERGY_CONSUMPTION 83
Total energy consumption for the GPU in mJ since the driver was last reloaded.
- #define NVML_FI_DEV_NVLINK_SPEED_MBPS_L0 84
NVLink Speed in MBps for Link 0.
- #define NVML_FI_DEV_NVLINK_SPEED_MBPS_L1 85
NVLink Speed in MBps for Link 1.
- #define NVML_FI_DEV_NVLINK_SPEED_MBPS_L2 86
NVLink Speed in MBps for Link 2.
- #define NVML_FI_DEV_NVLINK_SPEED_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_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L6 96
NVLink flow control CRC Error Counter for Lane 6.
- #define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L7 97
NVLink flow control CRC Error Counter for Lane 7.
- #define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L8 98

NVLink flow control CRC Error Counter for Lane 8.

- `#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L9` 99
NVLink flow control CRC Error Counter for Lane 9.
- `#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L10` 100
NVLink flow control CRC Error Counter for Lane 10.
- `#define NVML_FI_DEV_NVLINK_CRC_FLIT_ERROR_COUNT_L11` 101
NVLink flow control CRC Error Counter for Lane 11.
- `#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L6` 102
NVLink data CRC Error Counter for Lane 6.
- `#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L7` 103
NVLink data CRC Error Counter for Lane 7.
- `#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L8` 104
NVLink data CRC Error Counter for Lane 8.
- `#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L9` 105
NVLink data CRC Error Counter for Lane 9.
- `#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L10` 106
NVLink data CRC Error Counter for Lane 10.
- `#define NVML_FI_DEV_NVLINK_CRC_DATA_ERROR_COUNT_L11` 107
NVLink data CRC Error Counter for Lane 11.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L6` 108
NVLink Replay Error Counter for Lane 6.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L7` 109
NVLink Replay Error Counter for Lane 7.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L8` 110
NVLink Replay Error Counter for Lane 8.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L9` 111
NVLink Replay Error Counter for Lane 9.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L10` 112
NVLink Replay Error Counter for Lane 10.
- `#define NVML_FI_DEV_NVLINK_REPLAY_ERROR_COUNT_L11` 113
NVLink Replay Error Counter for Lane 11.
- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L6` 114
NVLink Recovery Error Counter for Lane 6.

- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L7` 115
NVLink Recovery Error Counter for Lane 7.
- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L8` 116
NVLink Recovery Error Counter for Lane 8.
- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L9` 117
NVLink Recovery Error Counter for Lane 9.
- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L10` 118
NVLink Recovery Error Counter for Lane 10.
- `#define NVML_FI_DEV_NVLINK_RECOVERY_ERROR_COUNT_L11` 119
NVLink Recovery Error Counter for Lane 11.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L6` 120
NVLink Bandwidth Counter for Counter Set 0, Lane 6.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L7` 121
NVLink Bandwidth Counter for Counter Set 0, Lane 7.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L8` 122
NVLink Bandwidth Counter for Counter Set 0, Lane 8.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L9` 123
NVLink Bandwidth Counter for Counter Set 0, Lane 9.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L10` 124
NVLink Bandwidth Counter for Counter Set 0, Lane 10.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C0_L11` 125
NVLink Bandwidth Counter for Counter Set 0, Lane 11.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L6` 126
NVLink Bandwidth Counter for Counter Set 1, Lane 6.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L7` 127
NVLink Bandwidth Counter for Counter Set 1, Lane 7.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L8` 128
NVLink Bandwidth Counter for Counter Set 1, Lane 8.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L9` 129
NVLink Bandwidth Counter for Counter Set 1, Lane 9.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L10` 130
NVLink Bandwidth Counter for Counter Set 1, Lane 10.
- `#define NVML_FI_DEV_NVLINK_BANDWIDTH_C1_L11` 131
NVLink Bandwidth Counter for Counter Set 1, Lane 11.

- `#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L6` 132
NVLink Speed in MBps for Link 6.
- `#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L7` 133
NVLink Speed in MBps for Link 7.
- `#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L8` 134
NVLink Speed in MBps for Link 8.
- `#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L9` 135
NVLink Speed in MBps for Link 9.
- `#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L10` 136
NVLink Speed in MBps for Link 10.
- `#define NVML_FI_DEV_NVLINK_SPEED_MBPS_L11` 137
NVLink Speed in MBps for Link 11.
- `#define NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_TX` 138
NVLink TX Data throughput in KiB.
- `#define NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_RX` 139
NVLink RX Data throughput in KiB.
- `#define NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_TX` 140
NVLink TX Data + protocol overhead in KiB.
- `#define NVML_FI_DEV_NVLINK_THROUGHPUT_RAW_RX` 141
NVLink RX Data + protocol overhead in KiB.
- `#define NVML_FI_DEV_REMAPPED_COR` 142
Number of remapped rows due to correctable errors.
- `#define NVML_FI_DEV_REMAPPED_UNC` 143
Number of remapped rows due to uncorrectable errors.
- `#define NVML_FI_DEV_REMAPPED_PENDING` 144
If any rows are pending remapping. 1=yes 0=no.
- `#define NVML_FI_DEV_REMAPPED_FAILURE` 145
If any rows failed to be remapped 1=yes 0=no.
- `#define NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID` 146
Remote device NVLink ID.
- `#define NVML_FI_MAX` 147
One greater than the largest field ID defined above.

6.6.1 Define Documentation

6.6.1.1 `#define NVML_FI_DEV_ECC_CURRENT 1`

Field Identifiers.

All Identifiers pertain to a device. Each ID is only used once and is guaranteed never to change.

6.6.1.2 `#define NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID 146`

Remote device NVLink ID

Link ID needs to be specified in the `scopeId` field in [nvmlFieldValue_t](#).

6.6.1.3 `#define NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_TX 138`

NVLink throughput counters field values

Link ID needs to be specified in the `scopeId` field in [nvmlFieldValue_t](#). A `scopeId` of `UINT_MAX` returns aggregate value summed up across all links for the specified counter type in `fieldId`.

6.7 Unit Structs

Data Structures

- struct [nvmlHwbcEntry_t](#)
- struct [nvmlLedState_t](#)
- struct [nvmlUnitInfo_t](#)
- struct [nvmlPSUInfo_t](#)
- struct [nvmlUnitFanInfo_t](#)
- struct [nvmlUnitFanSpeeds_t](#)

Enumerations

- enum [nvmlFanState_t](#) {
 [NVML_FAN_NORMAL](#) = 0,
 [NVML_FAN_FAILED](#) = 1 }
- enum [nvmlLedColor_t](#) {
 [NVML_LED_COLOR_GREEN](#) = 0,
 [NVML_LED_COLOR_AMBER](#) = 1 }

6.7.1 Enumeration Type Documentation

6.7.1.1 enum [nvmlFanState_t](#)

Fan state enum.

Enumerator:

NVML_FAN_NORMAL Fan is working properly.

NVML_FAN_FAILED Fan has failed.

6.7.1.2 enum [nvmlLedColor_t](#)

Led color enum.

Enumerator:

NVML_LED_COLOR_GREEN GREEN, indicates good health.

NVML_LED_COLOR_AMBER AMBER, indicates problem.

6.8 Event Types

Defines

- `#define nvmlEventTypeSingleBitEccError 0x0000000000000001LL`
Event about single bit ECC errors.
- `#define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL`
Event about double bit ECC errors.
- `#define nvmlEventTypePState 0x0000000000000004LL`
Event about PState changes.
- `#define nvmlEventTypeXidCriticalError 0x0000000000000008LL`
Event that Xid critical error occurred.
- `#define nvmlEventTypeClock 0x0000000000000010LL`
Event about clock changes.
- `#define nvmlEventTypePowerSourceChange 0x0000000000000080LL`
Event about AC/Battery power source changes.
- `#define nvmlEventMigConfigChange 0x0000000000000100LL`
Event about MIG configuration changes.
- `#define nvmlEventTypeNone 0x0000000000000000LL`
Mask with no events.
- `#define nvmlEventTypeAll`
Mask of all events.

6.8.1 Detailed Description

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

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

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

6.8.2 Define Documentation

6.8.2.1 `#define nvmlEventTypeClock 0x0000000000000010LL`

Kepler only

6.8.2.2 `#define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL`

Note:

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

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

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

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

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

6.9 Accounting Statistics

Data Structures

- struct [nvmlAccountingStats_t](#)

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetAccountingMode](#) (nvmlDevice_t device, [nvmlEnableState_t](#) *mode)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetAccountingStats](#) (nvmlDevice_t device, unsigned int pid, [nvmlAccountingStats_t](#) *stats)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetAccountingPids](#) (nvmlDevice_t device, unsigned int *count, unsigned int *pids)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetAccountingBufferSize](#) (nvmlDevice_t device, unsigned int *bufferSize)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetAccountingMode](#) (nvmlDevice_t device, [nvmlEnableState_t](#) mode)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceClearAccountingPids](#) (nvmlDevice_t device)

6.9.1 Detailed Description

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

Note:

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

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

6.9.2 Function Documentation

6.9.2.1 [nvmlReturn_t](#) DECLDIR [nvmlDeviceClearAccountingPids](#) (nvmlDevice_t device)

Clears accounting information about all processes that have already terminated.

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

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

Parameters:

device The identifier of the target device

Returns:

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

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

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

For Kepler TM or newer fully supported devices.

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

Parameters:

device The identifier of the target device

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

Returns:

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

See also:

[nvmlDeviceGetAccountingStats](#)

[nvmlDeviceGetAccountingPids](#)

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

Queries the state of per process accounting mode.

For Kepler TM or newer fully supported devices.

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

Parameters:

device The identifier of the target device

mode Reference in which to return the current accounting mode

Returns:

- [NVML_SUCCESS](#) if the mode has been successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *mode* are NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

For Kepler TM or newer fully supported devices.

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

For more details see [nvmlDeviceGetAccountingStats](#).

Note:

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

Parameters:

device The identifier of the target device

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

pids Reference in which to return list of process ids

Returns:

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

See also:

[nvmlDeviceGetAccountingBufferSize](#)

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

Queries process's accounting stats.

For Kepler TM or newer fully supported devices.

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

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

Note:

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

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

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

Warning:

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

Parameters:

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

Returns:

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

See also:

[nvmlDeviceGetAccountingBufferSize](#)

6.9.2.6 `nvmlReturn_t DECLDIR nvmlDeviceSetAccountingMode (nvmlDevice_t device, nvmlEnableState_t mode)`

Enables or disables per process accounting.

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

Note:

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

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

Disabling accounting clears all accounting pids information.

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

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

Parameters:

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

Returns:

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

6.10 Encoder Structs

Data Structures

- struct `nvmlEncoderSessionInfo_t`

Enumerations

- enum `nvmlEncoderType_t` {
 `NVML_ENCODER_QUERY_H264` = 0,
 `NVML_ENCODER_QUERY_HEVC` = 1 }

6.10.1 Enumeration Type Documentation

6.10.1.1 enum `nvmlEncoderType_t`

Represents type of encoder for capacity can be queried

Enumerator:

`NVML_ENCODER_QUERY_H264` H264 encoder.

`NVML_ENCODER_QUERY_HEVC` HEVC encoder.

6.11 Frame Buffer Capture Structures

Data Structures

- struct [nvmlFBCStats_t](#)
- struct [nvmlFBCSessionInfo_t](#)

Defines

- #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.

Enumerations

- enum [nvmlFBCSessionType_t](#) {
[NVML_FBC_SESSION_TYPE_UNKNOWN](#) = 0,
[NVML_FBC_SESSION_TYPE_TOSYS](#),
[NVML_FBC_SESSION_TYPE_CUDA](#),
[NVML_FBC_SESSION_TYPE_VID](#),
[NVML_FBC_SESSION_TYPE_HWENC](#) }

6.11.1 Enumeration Type Documentation

6.11.1.1 enum nvmlFBCSessionType_t

Represents frame buffer capture session type

Enumerator:

[NVML_FBC_SESSION_TYPE_UNKNOWN](#) Unknwon.
[NVML_FBC_SESSION_TYPE_TOSYS](#) ToSys.
[NVML_FBC_SESSION_TYPE_CUDA](#) Cuda.
[NVML_FBC_SESSION_TYPE_VID](#) Vid.
[NVML_FBC_SESSION_TYPE_HWENC](#) HEnc.

6.12 definitions related to the drain state

Enumerations

- enum [nvmlDetachGpuState_t](#)
- enum [nvmlPcieLinkState_t](#)

6.12.1 Enumeration Type Documentation

6.12.1.1 enum nvmlDetachGpuState_t

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

6.12.1.2 enum nvmlPcieLinkState_t

Parent bridge PCIe link state requested by nvmlDeviceRemoveGpu()

6.13 Initialization and Cleanup

Defines

- `#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.

Functions

- `nvmlReturn_t DECLDIR nvmlInit_v2 (void)`
- `nvmlReturn_t DECLDIR nvmlInitWithFlags (unsigned int flags)`
- `nvmlReturn_t DECLDIR nvmlShutdown (void)`

6.13.1 Detailed Description

This chapter describes the methods that handle NVML initialization and cleanup. It is the user's responsibility to call `nvmlInit_v2()` before calling any other methods, and `nvmlShutdown()` once NVML is no longer being used.

6.13.2 Function Documentation

6.13.2.1 `nvmlReturn_t DECLDIR nvmlInit_v2 (void)`

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

Note:

`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.

Note:

To contrast `nvmlInit_v2` with `nvmlInit_v1`, NVML 4.304 `nvmlInit_v1` will fail when any detected GPU is in a bad or unstable state.

For all products.

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

Returns:

- `NVML_SUCCESS` if NVML has been properly initialized
- `NVML_ERROR_DRIVER_NOT_LOADED` if NVIDIA driver is not running

- [NVML_ERROR_NO_PERMISSION](#) if NVML does not have permission to talk to the driver
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.13.2.2 `nvmlReturn_t DECLDIR nvmlInitWithFlags (unsigned int flags)`

`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_v2](#).

For all products.

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

6.13.2.3 `nvmlReturn_t DECLDIR nvmlShutdown (void)`

Shut down NVML by releasing all GPU resources previously allocated with [nvmlInit_v2\(\)](#).

For all products.

This method should be called after NVML work is done, once for each call to [nvmlInit_v2\(\)](#). A reference count of the number of initializations is maintained. Shutdown only occurs when the reference count reaches zero. For backwards compatibility, no error is reported if [nvmlShutdown\(\)](#) is called more times than `nvmlInit()`.

Returns:

- [NVML_SUCCESS](#) if NVML has been properly shut down
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.14 Error reporting

Functions

- `const DECLDIR char * nvmlErrorString (nvmlReturn_t result)`

6.14.1 Detailed Description

This chapter describes helper functions for error reporting routines.

6.14.2 Function Documentation

6.14.2.1 `const DECLDIR char* nvmlErrorString (nvmlReturn_t result)`

Helper method for converting NVML error codes into readable strings.

For all products.

Parameters:

result NVML error code to convert

Returns:

String representation of the error.

6.15 Constants

Defines

- `#define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE` 16
- `#define NVML_DEVICE_UUID_BUFFER_SIZE` 80
- `#define NVML_DEVICE_UUID_V2_BUFFER_SIZE` 96
- `#define NVML_DEVICE_PART_NUMBER_BUFFER_SIZE` 80
- `#define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE` 80
- `#define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE` 80
- `#define NVML_DEVICE_NAME_BUFFER_SIZE` 64
- `#define NVML_DEVICE_NAME_V2_BUFFER_SIZE` 96
- `#define NVML_DEVICE_SERIAL_BUFFER_SIZE` 30
- `#define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE` 32

6.15.1 Define Documentation

6.15.1.1 `#define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE` 16

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

6.15.1.2 `#define NVML_DEVICE_NAME_BUFFER_SIZE` 64

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

6.15.1.3 `#define NVML_DEVICE_NAME_V2_BUFFER_SIZE` 96

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

6.15.1.4 `#define NVML_DEVICE_PART_NUMBER_BUFFER_SIZE` 80

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

6.15.1.5 `#define NVML_DEVICE_SERIAL_BUFFER_SIZE` 30

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

6.15.1.6 `#define NVML_DEVICE_UUID_BUFFER_SIZE` 80

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

6.15.1.7 `#define NVML_DEVICE_UUID_V2_BUFFER_SIZE` 96

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

6.15.1.8 #define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32

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

6.15.1.9 #define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80

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

6.15.1.10 #define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80

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

6.16 System Queries

Defines

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

Functions

- `nvmlReturn_t DECLDIR nvmlSystemGetDriverVersion` (char *version, unsigned int length)
- `nvmlReturn_t DECLDIR nvmlSystemGetNVMLVersion` (char *version, unsigned int length)
- `nvmlReturn_t DECLDIR nvmlSystemGetCudaDriverVersion` (int *cudaDriverVersion)
- `nvmlReturn_t DECLDIR nvmlSystemGetCudaDriverVersion_v2` (int *cudaDriverVersion)
- `nvmlReturn_t DECLDIR nvmlSystemGetProcessName` (unsigned int pid, char *name, unsigned int length)

6.16.1 Detailed Description

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

6.16.2 Define Documentation

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

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

6.16.3 Function Documentation

6.16.3.1 `nvmlReturn_t DECLDIR nvmlSystemGetCudaDriverVersion` (int * *cudaDriverVersion*)

Retrieves the version of the CUDA driver.

For all products.

The CUDA driver version returned will be retrieved from the currently installed version of CUDA. If the cuda library is not found, this function will return a known supported version number.

Parameters:

cudaDriverVersion Reference in which to return the version identifier

Returns:

- `NVML_SUCCESS` if *cudaDriverVersion* has been set
- `NVML_ERROR_INVALID_ARGUMENT` if *cudaDriverVersion* is NULL

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

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

For all products.

The returned CUDA driver version by calling `cuDriverGetVersion()`

Parameters:

cudaDriverVersion Reference in which to return the version identifier

Returns:

- [NVML_SUCCESS](#) if *cudaDriverVersion* has been set
- [NVML_ERROR_INVALID_ARGUMENT](#) if *cudaDriverVersion* is NULL
- [NVML_ERROR_LIBRARY_NOT_FOUND](#) if *libcuda.so.1* or *libcuda.dll* is not found
- [NVML_ERROR_FUNCTION_NOT_FOUND](#) if *cuDriverGetVersion()* is not found in the shared library

6.16.3.3 nvmlReturn_t DECLDIR nvmlSystemGetDriverVersion (char * *version*, unsigned int *length*)

Retrieves the version of the system's graphics driver.

For all products.

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

Parameters:

version Reference in which to return the version identifier

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

Returns:

- [NVML_SUCCESS](#) if *version* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *version* is NULL
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *length* is too small

6.16.3.4 nvmlReturn_t DECLDIR nvmlSystemGetNVMLVersion (char * *version*, unsigned int *length*)

Retrieves the version of the NVML library.

For all products.

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

Parameters:

version Reference in which to return the version identifier

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

Returns:

- [NVML_SUCCESS](#) if *version* has been set
- [NVML_ERROR_INVALID_ARGUMENT](#) if *version* is NULL
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *length* is too small

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

Gets name of the process with provided process id

For all products.

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

Parameters:

pid The identifier of the process

name Reference in which to return the process name

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

Returns:

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

6.17 Unit Queries

Functions

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

6.17.1 Detailed Description

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

6.17.2 Function Documentation

6.17.2.1 [nvmlReturn_t](#) DECLDIR [nvmlSystemGetHicVersion](#) (unsigned int * *hwbcCount*, [nvmlHwbcEntry_t](#) * *hwbcEntries*)

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

For S-class products.

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

Parameters:

hwbcCount Size of hwbcEntries array

hwbcEntries Array holding information about hwbc

Returns:

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

6.17.2.2 [nvmlReturn_t](#) DECLDIR [nvmlUnitGetCount](#) (unsigned int * *unitCount*)

Retrieves the number of units in the system.

For S-class products.

Parameters:

unitCount Reference in which to return the number of units

Returns:

- [NVML_SUCCESS](#) if *unitCount* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *unitCount* is NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

For S-class products.

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

Parameters:

unit The identifier of the target unit

deviceCount Reference in which to provide the *devices* array size, and to return the number of attached GPU devices

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

Returns:

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

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

Retrieves the fan speed readings for the unit.

For S-class products.

See [nvmlUnitFanSpeeds_t](#) for details on available fan speed info.

Parameters:

unit The identifier of the target unit

fanSpeeds Reference in which to return the fan speed information

Returns:

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

6.17.2.5 `nvmlReturn_t DECLDIR nvmlUnitGetHandleByIndex (unsigned int index, nvmlUnit_t * unit)`

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

For S-class products.

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

The order in which NVML enumerates units has no guarantees of consistency between reboots.

Parameters:

index The index of the target unit, ≥ 0 and $< unitCount$

unit Reference in which to return the unit handle

Returns:

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

6.17.2.6 `nvmlReturn_t DECLDIR nvmlUnitGetLedState (nvmlUnit_t unit, nvmlLedState_t * state)`

Retrieves the LED state associated with this unit.

For S-class products.

See [nvmlLedState_t](#) for details on allowed states.

Parameters:

unit The identifier of the target unit

state Reference in which to return the current LED state

Returns:

- [NVML_SUCCESS](#) if *state* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *unit* is invalid or *state* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if this is not an S-class product
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlUnitSetLedState\(\)](#)

6.17.2.7 `nvmlReturn_t DECLDIR nvmlUnitGetPsuInfo (nvmlUnit_t unit, nvmlPSUInfo_t * psu)`

Retrieves the PSU stats for the unit.

For S-class products.

See [nvmlPSUInfo_t](#) for details on available PSU info.

Parameters:

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

Returns:

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

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

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

For S-class products.

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

Parameters:

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

Returns:

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

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

Retrieves the static information associated with a unit.

For S-class products.

See [nvmlUnitInfo_t](#) for details on available unit info.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if *info* has been populated
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *unit* is invalid or *info* is NULL

6.18 Device Queries

Modules

- [CPU and Memory Affinity](#)

Functions

- [nvmlReturn_t DECLDIR nvmlDeviceGetCount_v2](#) (unsigned int *deviceCount)
- [nvmlReturn_t DECLDIR nvmlDeviceGetAttributes_v2](#) (nvmlDevice_t device, nvmlDeviceAttributes_t *attributes)
- [nvmlReturn_t DECLDIR nvmlDeviceGetHandleByIndex_v2](#) (unsigned int index, nvmlDevice_t *device)
- [nvmlReturn_t DECLDIR nvmlDeviceGetHandleBySerial](#) (const char *serial, nvmlDevice_t *device)
- [nvmlReturn_t DECLDIR nvmlDeviceGetHandleByUUID](#) (const char *uuid, nvmlDevice_t *device)
- [nvmlReturn_t DECLDIR nvmlDeviceGetHandleByPciBusId_v2](#) (const char *pciBusId, nvmlDevice_t *device)
- [nvmlReturn_t DECLDIR nvmlDeviceGetName](#) (nvmlDevice_t device, char *name, unsigned int length)
- [nvmlReturn_t DECLDIR nvmlDeviceGetBrand](#) (nvmlDevice_t device, [nvmlBrandType_t](#) *type)
- [nvmlReturn_t DECLDIR nvmlDeviceGetIndex](#) (nvmlDevice_t device, unsigned int *index)
- [nvmlReturn_t DECLDIR nvmlDeviceGetSerial](#) (nvmlDevice_t device, char *serial, unsigned int length)
- [nvmlReturn_t DECLDIR nvmlDeviceGetTopologyCommonAncestor](#) (nvmlDevice_t device1, nvmlDevice_t device2, [nvmlGpuTopologyLevel_t](#) *pathInfo)
- [nvmlReturn_t DECLDIR nvmlDeviceGetTopologyNearestGpus](#) (nvmlDevice_t device, [nvmlGpuTopologyLevel_t](#) level, unsigned int *count, nvmlDevice_t *deviceArray)
- [nvmlReturn_t DECLDIR nvmlSystemGetTopologyGpuSet](#) (unsigned int cpuNumber, unsigned int *count, nvmlDevice_t *deviceArray)
- [nvmlReturn_t DECLDIR nvmlDeviceGetP2PStatus](#) (nvmlDevice_t device1, nvmlDevice_t device2, [nvmlGpuP2PCapsIndex_t](#) p2pIndex, [nvmlGpuP2PStatus_t](#) *p2pStatus)
- [nvmlReturn_t DECLDIR nvmlDeviceGetUUID](#) (nvmlDevice_t device, char *uuid, unsigned int length)
- [nvmlReturn_t DECLDIR nvmlVgpuInstanceGetMdevUUID](#) (nvmlVgpuInstance_t vgpuInstance, char *mdevUuid, unsigned int size)
- [nvmlReturn_t DECLDIR nvmlDeviceGetMinorNumber](#) (nvmlDevice_t device, unsigned int *minorNumber)
- [nvmlReturn_t DECLDIR nvmlDeviceGetBoardPartNumber](#) (nvmlDevice_t device, char *partNumber, unsigned int length)
- [nvmlReturn_t DECLDIR nvmlDeviceGetInforomVersion](#) (nvmlDevice_t device, [nvmlInforomObject_t](#) object, char *version, unsigned int length)
- [nvmlReturn_t DECLDIR nvmlDeviceGetInforomImageVersion](#) (nvmlDevice_t device, char *version, unsigned int length)
- [nvmlReturn_t DECLDIR nvmlDeviceGetInforomConfigurationChecksum](#) (nvmlDevice_t device, unsigned int *checksum)
- [nvmlReturn_t DECLDIR nvmlDeviceValidateInforom](#) (nvmlDevice_t device)
- [nvmlReturn_t DECLDIR nvmlDeviceGetDisplayMode](#) (nvmlDevice_t device, [nvmlEnableState_t](#) *display)
- [nvmlReturn_t DECLDIR nvmlDeviceGetDisplayActive](#) (nvmlDevice_t device, [nvmlEnableState_t](#) *isActive)
- [nvmlReturn_t DECLDIR nvmlDeviceGetPersistenceMode](#) (nvmlDevice_t device, [nvmlEnableState_t](#) *mode)
- [nvmlReturn_t DECLDIR nvmlDeviceGetPciInfo_v3](#) (nvmlDevice_t device, [nvmlPciInfo_t](#) *pci)
- [nvmlReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkGeneration](#) (nvmlDevice_t device, unsigned int *maxLinkGen)
- [nvmlReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkWidth](#) (nvmlDevice_t device, unsigned int *maxLinkWidth)
- [nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkGeneration](#) (nvmlDevice_t device, unsigned int *currLinkGen)

- `nvmlReturn_t` DECLDIR `nvmlDeviceGetCurrPcieLinkWidth` (`nvmlDevice_t` device, unsigned int *currLinkWidth)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPcieThroughput` (`nvmlDevice_t` device, `nvmlPcieUtilCounter_t` counter, unsigned int *value)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPcieReplayCounter` (`nvmlDevice_t` device, unsigned int *value)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetClockInfo` (`nvmlDevice_t` device, `nvmlClockType_t` type, unsigned int *clock)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetMaxClockInfo` (`nvmlDevice_t` device, `nvmlClockType_t` type, unsigned int *clock)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetApplicationsClock` (`nvmlDevice_t` device, `nvmlClockType_t` clockType, unsigned int *clockMHz)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetDefaultApplicationsClock` (`nvmlDevice_t` device, `nvmlClockType_t` clockType, unsigned int *clockMHz)
- `nvmlReturn_t` DECLDIR `nvmlDeviceResetApplicationsClocks` (`nvmlDevice_t` device)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetClock` (`nvmlDevice_t` device, `nvmlClockType_t` clockType, `nvmlClockId_t` clockId, unsigned int *clockMHz)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetMaxCustomerBoostClock` (`nvmlDevice_t` device, `nvmlClockType_t` clockType, unsigned int *clockMHz)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetSupportedMemoryClocks` (`nvmlDevice_t` device, unsigned int *count, unsigned int *clocksMHz)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetSupportedGraphicsClocks` (`nvmlDevice_t` device, unsigned int memoryClockMHz, unsigned int *count, unsigned int *clocksMHz)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetAutoBoostedClocksEnabled` (`nvmlDevice_t` device, `nvmlEnableState_t` *isEnabled, `nvmlEnableState_t` *defaultIsEnabled)
- `nvmlReturn_t` DECLDIR `nvmlDeviceSetAutoBoostedClocksEnabled` (`nvmlDevice_t` device, `nvmlEnableState_t` enabled)
- `nvmlReturn_t` DECLDIR `nvmlDeviceSetDefaultAutoBoostedClocksEnabled` (`nvmlDevice_t` device, `nvmlEnableState_t` enabled, unsigned int flags)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetFanSpeed` (`nvmlDevice_t` device, unsigned int *speed)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetFanSpeed_v2` (`nvmlDevice_t` device, unsigned int fan, unsigned int *speed)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetTemperature` (`nvmlDevice_t` device, `nvmlTemperatureSensors_t` sensorType, unsigned int *temp)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetTemperatureThreshold` (`nvmlDevice_t` device, `nvmlTemperatureThresholds_t` thresholdType, unsigned int *temp)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPerformanceState` (`nvmlDevice_t` device, `nvmlPstates_t` *pState)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetCurrentClocksThrottleReasons` (`nvmlDevice_t` device, unsigned long long *clocksThrottleReasons)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetSupportedClocksThrottleReasons` (`nvmlDevice_t` device, unsigned long long *supportedClocksThrottleReasons)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPowerState` (`nvmlDevice_t` device, `nvmlPstates_t` *pState)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPowerManagementMode` (`nvmlDevice_t` device, `nvmlEnableState_t` *mode)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPowerManagementLimit` (`nvmlDevice_t` device, unsigned int *limit)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPowerManagementLimitConstraints` (`nvmlDevice_t` device, unsigned int *minLimit, unsigned int *maxLimit)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPowerManagementDefaultLimit` (`nvmlDevice_t` device, unsigned int *defaultLimit)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetPowerUsage` (`nvmlDevice_t` device, unsigned int *power)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetTotalEnergyConsumption` (`nvmlDevice_t` device, unsigned long long *energy)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetEnforcedPowerLimit` (`nvmlDevice_t` device, unsigned int *limit)

- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetGpuOperationMode](#) (nvmlDevice_t device, [nvmlGpuOperationMode_t](#) *current, [nvmlGpuOperationMode_t](#) *pending)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetMemoryInfo](#) (nvmlDevice_t device, [nvmlMemory_t](#) *memory)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetComputeMode](#) (nvmlDevice_t device, [nvmlComputeMode_t](#) *mode)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetCudaComputeCapability](#) (nvmlDevice_t device, int *major, int *minor)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetEccMode](#) (nvmlDevice_t device, [nvmlEnableState_t](#) *current, [nvmlEnableState_t](#) *pending)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetBoardId](#) (nvmlDevice_t device, unsigned int *boardId)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetMultiGpuBoard](#) (nvmlDevice_t device, unsigned int *multiGpuBool)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetTotalEccErrors](#) (nvmlDevice_t device, [nvmlMemoryErrorType_t](#) errorType, [nvmlEccCounterType_t](#) counterType, unsigned long long *eccCounts)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetDetailedEccErrors](#) (nvmlDevice_t device, [nvmlMemoryErrorType_t](#) errorType, [nvmlEccCounterType_t](#) counterType, [nvmlEccErrorCounts_t](#) *eccCounts)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetMemoryErrorCounter](#) (nvmlDevice_t device, [nvmlMemoryErrorType_t](#) errorType, [nvmlEccCounterType_t](#) counterType, [nvmlMemoryLocation_t](#) locationType, unsigned long long *count)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetUtilizationRates](#) (nvmlDevice_t device, [nvmlUtilization_t](#) *utilization)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetEncoderUtilization](#) (nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetEncoderCapacity](#) (nvmlDevice_t device, [nvmlEncoderType_t](#) encoderQueryType, unsigned int *encoderCapacity)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetEncoderStats](#) (nvmlDevice_t device, unsigned int *sessionCount, unsigned int *averageFps, unsigned int *averageLatency)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetEncoderSessions](#) (nvmlDevice_t device, unsigned int *sessionCount, [nvmlEncoderSessionInfo_t](#) *sessionInfos)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetDecoderUtilization](#) (nvmlDevice_t device, unsigned int *utilization, unsigned int *samplingPeriodUs)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetFBCStats](#) (nvmlDevice_t device, [nvmlFBCStats_t](#) *fbcStats)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetFBCSessions](#) (nvmlDevice_t device, unsigned int *sessionCount, [nvmlFBCSessionInfo_t](#) *sessionInfo)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetDriverModel](#) (nvmlDevice_t device, [nvmlDriverModel_t](#) *current, [nvmlDriverModel_t](#) *pending)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetVbiosVersion](#) (nvmlDevice_t device, char *version, unsigned int length)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetBridgeChipInfo](#) (nvmlDevice_t device, [nvmlBridgeChipHierarchy_t](#) *bridgeHierarchy)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetComputeRunningProcesses_v2](#) (nvmlDevice_t device, unsigned int *infoCount, [nvmlProcessInfo_t](#) *infos)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetGraphicsRunningProcesses_v2](#) (nvmlDevice_t device, unsigned int *infoCount, [nvmlProcessInfo_t](#) *infos)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceOnSameBoard](#) (nvmlDevice_t device1, nvmlDevice_t device2, int *onSameBoard)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetAPIRestriction](#) (nvmlDevice_t device, [nvmlRestrictedAPI_t](#) apiType, [nvmlEnableState_t](#) *isRestricted)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetSamples](#) (nvmlDevice_t device, [nvmlSamplingType_t](#) type, unsigned long long lastSeenTimeStamp, [nvmlValueType_t](#) *sampleValType, unsigned int *sampleCount, [nvmlSample_t](#) *samples)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetBAR1MemoryInfo](#) (nvmlDevice_t device, [nvmlBAR1Memory_t](#) *bar1Memory)

- `nvmlReturn_t` DECLDIR `nvmlDeviceGetViolationStatus` (`nvmlDevice_t` device, `nvmlPerfPolicyType_t` perfPolicyType, `nvmlViolationTime_t` *violTime)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetRetiredPages` (`nvmlDevice_t` device, `nvmlPageRetirementCause_t` cause, unsigned int *pageCount, unsigned long long *addresses)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetRetiredPages_v2` (`nvmlDevice_t` device, `nvmlPageRetirementCause_t` cause, unsigned int *pageCount, unsigned long long *addresses, unsigned long long *timestamps)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetRetiredPagesPendingStatus` (`nvmlDevice_t` device, `nvmlEnableState_t` *isPending)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetRemappedRows` (`nvmlDevice_t` device, unsigned int *corrRows, unsigned int *uncRows, unsigned int *isPending, unsigned int *failureOccurred)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetRowRemapperHistogram` (`nvmlDevice_t` device, `nvmlRowRemapperHistogramValues_t` *values)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetArchitecture` (`nvmlDevice_t` device, `nvmlDeviceArchitecture_t` *arch)

6.18.1 Detailed Description

This chapter describes that queries that NVML can perform against each device. In each case the device is identified with an `nvmlDevice_t` handle. This handle is obtained by calling one of `nvmlDeviceGetHandleByIndex_v2()`, `nvmlDeviceGetHandleBySerial()`, `nvmlDeviceGetHandleByPciBusId_v2()`, or `nvmlDeviceGetHandleByUUID()`.

6.18.2 Function Documentation

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

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

For all fully supported products.

Parameters:

device The identifier of the target device

apiType Target API type for this operation

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

Returns:

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

See also:

`nvmlRestrictedAPI_t`

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

Retrieves the current setting of a clock that applications will use unless an overspec situation occurs. Can be changed using [nvmlDeviceSetApplicationsClocks](#).

For Kepler TM or newer fully supported devices.

Parameters:

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

Returns:

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

6.18.2.3 `nvmlReturn_t DECLDIR nvmlDeviceGetArchitecture (nvmlDevice_t device, nvmlDeviceArchitecture_t * arch)`

Get architecture for device

Parameters:

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

Returns:

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

6.18.2.4 `nvmlReturn_t DECLDIR nvmlDeviceGetAttributes_v2 (nvmlDevice_t device, nvmlDeviceAttributes_t * attributes)`

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

Note:

This API currently only supports MIG device handles.

For newer than Volta TM fully supported devices. Supported on Linux only.

Parameters:

device NVML device handle

attributes Device attributes

Returns:

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

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

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

For Kepler™ or newer fully supported devices.

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

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

Parameters:

device The identifier of the target device

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

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

Returns:

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

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

Gets Total, Available and Used size of BAR1 memory.

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

Note:

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

For Kepler TM or newer fully supported devices.

Parameters:

device The identifier of the target device

bar1Memory Reference in which BAR1 memory information is returned.

Returns:

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

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

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

For Fermi TM or newer fully supported devices.

Parameters:

device The identifier of the target device

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

Returns:

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

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

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

For all products.

Parameters:

device Identifier of the target device

partNumber Reference to the buffer to return

length Length of the buffer reference

Returns:

- [NVML_SUCCESS](#) if *partNumber* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_NOT_SUPPORTED](#) if the needed VBIOS fields have not been filled
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *serial* is NULL
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.18.2.9 `nvmlReturn_t DECLDIR nvmlDeviceGetBrand (nvmlDevice_t device, nvmlBrandType_t * type)`

Retrieves the brand of this device.

For all products.

The type is a member of [nvmlBrandType_t](#) defined above.

Parameters:

device The identifier of the target device

type Reference in which to return the product brand type

Returns:

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

6.18.2.10 `nvmlReturn_t DECLDIR nvmlDeviceGetBridgeChipInfo (nvmlDevice_t device, nvmlBridgeChipHierarchy_t * bridgeHierarchy)`

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

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

Parameters:

device The identifier of the target device

bridgeHierarchy Reference to the returned bridge chip Hierarchy

Returns:

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

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

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

For Kepler TM or newer fully supported devices.

Parameters:

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

Returns:

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

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

Retrieves the current clock speeds for the device.

For Fermi TM or newer fully supported devices.

See [nvmlClockType_t](#) for details on available clock information.

Parameters:

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

Returns:

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

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

Retrieves the current compute mode for the device.

For all products.

See [nvmlComputeMode_t](#) for details on allowed compute modes.

Parameters:

device The identifier of the target device

mode Reference in which to return the current compute mode

Returns:

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

See also:

[nvmlDeviceSetComputeMode\(\)](#)

6.18.2.14 `nvmlReturn_t DECLDIR nvmlDeviceGetComputeRunningProcesses_v2 (nvmlDevice_t device, unsigned int * infoCount, nvmlProcessInfo_t * infos)`

Get information about processes with a compute context on a device

For Fermi TM or newer fully supported devices.

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

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

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

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

Note:

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

Parameters:

device The device handle or MIG device handle

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

infos Reference in which to return the process information

Returns:

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

See also:

[nvmlSystemGetProcessName](#)

6.18.2.15 `nvmlReturn_t DECLDIR nvmlDeviceGetCount_v2 (unsigned int * deviceCount)`

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

For all products.

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

Parameters:

deviceCount Reference in which to return the number of accessible devices

Returns:

- [NVML_SUCCESS](#) if *deviceCount* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *deviceCount* is NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Retrieves the CUDA compute capability of the device.

For all products.

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

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

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

Retrieves current clocks throttling reasons.

For all fully supported products.

Note:

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

Parameters:

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

Returns:

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

See also:

[NvmlClocksThrottleReasons](#)
[nvmlDeviceGetSupportedClocksThrottleReasons](#)

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

Retrieves the current PCIe link generation

For Fermi TM or newer fully supported devices.

Parameters:

device The identifier of the target device

currLinkGen Reference in which to return the current PCIe link generation

Returns:

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

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

Retrieves the current PCIe link width

For Fermi TM or newer fully supported devices.

Parameters:

device The identifier of the target device

currLinkWidth Reference in which to return the current PCIe link generation

Returns:

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

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

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

For Kepler TM or newer fully supported devices.

Note:

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

Parameters:

device The identifier of the target device

utilization Reference to an unsigned int for decoder utilization info

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

Returns:

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

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

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

For Kepler TM or newer fully supported devices.

Parameters:

device The identifier of the target device

clockType Identify which clock domain to query

clockMHz Reference in which to return the default clock in MHz

Returns:

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

See also:

[nvmlDeviceGetApplicationsClock](#)

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

Retrieves the detailed ECC error counts for the device.

Deprecated

This API supports only a fixed set of ECC error locations. On different GPU architectures different locations are supported. See [nvmlDeviceGetMemoryErrorCounter](#)

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

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

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

See [nvmlMemoryErrorType_t](#) for a description of available bit types.

See [nvmlEccCounterType_t](#) for a description of available counter types.

See [nvmlEccErrorCounts_t](#) for a description of provided detailed ECC counts.

Parameters:

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

Returns:

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

See also:

[nvmlDeviceClearEccErrorCounts\(\)](#)

6.18.2.23 [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetDisplayActive](#) ([nvmlDevice_t](#) *device*, [nvmlEnableState_t](#) * *isActive*)

Retrieves the display active state for the device.

For all products.

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

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

See [nvmlEnableState_t](#) for details on allowed modes.

Parameters:

- device* The identifier of the target device
- isActive* Reference in which to return the display active state

Returns:

- [NVML_SUCCESS](#) if *isActive* has been set

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

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

Retrieves the display mode for the device.

For all products.

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

See [nvmlEnableState_t](#) for details on allowed modes.

Parameters:

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

Returns:

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

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

Retrieves the current and pending driver model for the device.

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

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

See [nvmlDriverModel_t](#) for details on available driver models.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if either *current* and/or *pending* have been set

- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or both *current* and *pending* are NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the platform is not windows
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceSetDriverModel\(\)](#)

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

Retrieves the current and pending ECC modes for the device.

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

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

See [nvmlEnableState_t](#) for details on allowed modes.

Parameters:

device The identifier of the target device

current Reference in which to return the current ECC mode

pending Reference in which to return the pending ECC mode

Returns:

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

See also:

[nvmlDeviceSetEccMode\(\)](#)

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

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

For Maxwell TM or newer fully supported devices.

Parameters:

device The identifier of the target device

encoderQueryType Type of encoder to query

encoderCapacity Reference to an unsigned int for the encoder capacity

Returns:

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

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

Retrieves information about active encoder sessions on a target device.

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

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

For Maxwell™ or newer fully supported devices.

Parameters:

device The identifier of the target device

sessionCount Reference to caller supplied array size, and returns the number of sessions.

sessionInfos Reference in which to return the session information

Returns:

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

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

Retrieves the current encoder statistics for a given device.

For Maxwell™ or newer fully supported devices.

Parameters:

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

Returns:

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

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

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

For Kepler TM or newer fully supported devices.

Note:

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

Parameters:

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

Returns:

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

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

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

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

For Kepler TM or newer fully supported devices.

Parameters:

device The device to communicate with

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

Returns:

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

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

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

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

For all discrete products with dedicated fans.

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

Parameters:

device The identifier of the target device

speed Reference in which to return the fan speed percentage

Returns:

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

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

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

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

For all discrete products with dedicated fans.

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

Parameters:

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

Returns:

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

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

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

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

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

For Maxwell TM or newer fully supported devices.

Note:

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

Parameters:

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

Returns:

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

6.18.2.35 `nvmlReturn_t DECLDIR nvmlDeviceGetFBCStats (nvmlDevice_t device, nvmlFBCStats_t * fbcStats)`

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

For Maxwell TM or newer fully supported devices.

Parameters:

device The identifier of the target device

fbcStats Reference to `nvmlFBCStats_t` structure containing NvFBC stats

Returns:

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

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

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

For GK110 M-class and X-class Tesla TM 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 TM C-class products.

Parameters:

device The identifier of the target device

current Reference in which to return the current GOM

pending Reference in which to return the pending GOM

Returns:

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

See also:

`nvmlGpuOperationMode_t`
`nvmlDeviceSetGpuOperationMode`

6.18.2.37 `nvmlReturn_t DECLDIR nvmlDeviceGetGraphicsRunningProcesses_v2 (nvmlDevice_t device, unsigned int * infoCount, nvmlProcessInfo_t * infos)`

Get information about processes with a graphics context on a device

For Kepler™ or newer fully supported devices.

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

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

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

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

Note:

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

Parameters:

device The 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_NO_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML_ERROR_INVALID_ARGUMENT](#) if `device` is invalid, either of `infoCount` or `infos` is `NULL`
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_NOT_SUPPORTED](#) if this query is not supported by `device`
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlSystemGetProcessName](#)

6.18.2.38 `nvmlReturn_t DECLDIR nvmlDeviceGetHandleByIndex_v2 (unsigned int index, nvmlDevice_t * device)`

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

For all products.

Valid indices are derived from the `accessibleDevices` count returned by [nvmlDeviceGetCount_v2\(\)](#). For example, if `accessibleDevices` is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

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

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

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

- The target GPU is an SLI slave

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

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

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

See also:

[nvmlDeviceGetIndex](#)

[nvmlDeviceGetCount](#)

6.18.2.39 `nvmlReturn_t DECLDIR nvmlDeviceGetHandleByPciBusId_v2 (const char * pciBusId, nvmlDevice_t * device)`

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

For all products.

This value corresponds to the [nvmlPciInfo_t::busId](#) returned by [nvmlDeviceGetPciInfo_v3\(\)](#).

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

- The target GPU is an SLI slave

Note:

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

Parameters:

pciBusId The PCI bus id of the target GPU

device Reference in which to return the device handle

Returns:

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

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

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

For Fermi TM or newer fully supported devices.

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

Deprecated

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

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

Parameters:

serial The board serial number of the target GPU

device Reference in which to return the device handle

Returns:

- `NVML_SUCCESS` if *device* has been set
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if *serial* is invalid, *device* is NULL or more than one device has the same serial (dual GPU boards)

- [NVML_ERROR_NOT_FOUND](#) if *serial* does not match a valid device on the system
- [NVML_ERROR_INSUFFICIENT_POWER](#) if any attached devices have improperly attached external power cables
- [NVML_ERROR_IRQ_ISSUE](#) if NVIDIA kernel detected an interrupt issue with the attached GPUs
- [NVML_ERROR_GPU_IS_LOST](#) if any GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

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

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

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

Parameters:

uuid The UUID of the target GPU
device Reference in which to return the device handle

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

This API does not currently support acquiring MIG device handles using MIG device UUIDs.

Returns:

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

See also:

[nvmlDeviceGetUUID](#)

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

Retrieves the NVML index of this device.

For all products.

Valid indices are derived from the *accessibleDevices* count returned by [nvmlDeviceGetCount_v2\(\)](#). For example, if *accessibleDevices* is 2 the valid indices are 0 and 1, corresponding to GPU 0 and GPU 1.

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

When used with MIG device handles this API returns indices that can be passed to [nvmlDeviceGetMigDeviceHandleByIndex](#) to retrieve an identical handle. MIG device indices are unique within a device.

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

Parameters:

- device* The identifier of the target device
- index* Reference in which to return the NVML index of the device

Returns:

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

See also:

[nvmlDeviceGetHandleByIndex\(\)](#)
[nvmlDeviceGetCount\(\)](#)

6.18.2.43 [nvmlReturn_t DECLDIR nvmlDeviceGetInforomConfigurationChecksum \(nvmlDevice_t device, unsigned int * checksum\)](#)

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

For all products with an inforom.

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

Parameters:

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

Returns:

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

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

Retrieves the global infoROM image version

For all products with an inforom.

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

Parameters:

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

Returns:

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

See also:

[nvmlDeviceGetInforomVersion](#)

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

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

For all products with an inforom.

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

See [nvmlInforomObject_t](#) for details on the available infoROM objects.

Parameters:

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

Returns:

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

See also:

[nvmlDeviceGetInforomImageVersion](#)

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

Retrieves the maximum clock speeds for the device.

For Fermi [™] or newer fully supported devices.

See [nvmlClockType_t](#) for details on available clock information.

Note:

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

Parameters:

device The identifier of the target device

type Identify which clock domain to query

clock Reference in which to return the clock speed in MHz

Returns:

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

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

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

For Pascal [™] or newer fully supported devices.

Parameters:

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

Returns:

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

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

Retrieves the maximum PCIe link generation possible with this device and system

I.E. for a generation 2 PCIe device attached to a generation 1 PCIe bus the max link generation this function will report is generation 1.

For Fermi TM or newer fully supported devices.

Parameters:

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

Returns:

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

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

Retrieves the maximum PCIe link width possible with this device and system

I.E. for a device with a 16x PCIe bus width attached to a 8x PCIe system bus this function will report a max link width of 8.

For Fermi TM or newer fully supported devices.

Parameters:

device The identifier of the target device

maxLinkWidth Reference in which to return the max PCIe link generation

Returns:

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

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

Retrieves the requested memory error counter for the device.

For Fermi TM or newer fully supported devices. Requires *NVML_INFOROM_ECC* version 2.0 or higher to report aggregate location-based memory error counts. Requires *NVML_INFOROM_ECC* version 1.0 or higher to report all other memory error counts.

Only applicable to devices with ECC.

Requires ECC Mode to be enabled.

Note:

On MIG-enabled GPUs, per instance information can be queried using specific MIG device handles. Per instance information is currently only supported for non-DRAM uncorrectable volatile errors. Querying volatile errors using device handles is currently not supported.

See [nvmlMemoryErrorType_t](#) for a description of available memory error types.

See [nvmlEccCounterType_t](#) for a description of available counter types.

See [nvmlMemoryLocation_t](#) for a description of available counter locations.

Parameters:

device The identifier of the target device
errorType Flag that specifies the type of error.
counterType Flag that specifies the counter-type of the errors.
locationType Specifies the location of the counter.
count Reference in which to return the ECC counter

Returns:

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

6.18.2.51 `nvmlReturn_t DECLDIR nvmlDeviceGetMemoryInfo (nvmlDevice_t device, nvmlMemory_t * memory)`

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

For all products.

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

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

See [nvmlMemory_t](#) for details on available memory info.

Note:

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

Parameters:

device The identifier of the target device

memory Reference in which to return the memory information

Returns:

- [NVML_SUCCESS](#) if *memory* has been populated
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_NO_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *memory* is NULL
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

For all products. Supported only for Linux

Parameters:

device The identifier of the target device

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

Returns:

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

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

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

For Fermi™ or newer fully supported devices.

Parameters:

device The identifier of the target device

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

Returns:

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

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

Retrieves the name of this device.

For all products.

The name is an alphanumeric string that denotes a particular product, e.g. Tesla™C2070. It will not exceed 96 characters in length (including the NULL terminator). See [nvmlConstants::NVML_DEVICE_NAME_V2_BUFFER_SIZE](#).

When used with MIG device handles the API returns MIG device names which can be used to identify devices based on their attributes.

Parameters:

device The identifier of the target device

name Reference in which to return the product name

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

Returns:

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

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

Retrieve the status for a given p2p capability index between a given pair of GPU

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

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

Retrieve the PCIe replay counter.

For Kepler TM or newer fully supported devices.

Parameters:

device The identifier of the target device

value Reference in which to return the counter's value

Returns:

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

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

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

For Maxwell TM or newer fully supported devices.

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

Parameters:

- device* The identifier of the target device
- counter* The specific counter that should be queried [nvmlPcieUtilCounter_t](#)
- value* Reference in which to return throughput in KB/s

Returns:

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

6.18.2.58 nvmlReturn_t DECLDIR nvmlDeviceGetPciInfo_v3 (nvmlDevice_t device, nvmlPciInfo_t * pci)

Retrieves the PCI attributes of this device.

For all products.

See [nvmlPciInfo_t](#) for details on the available PCI info.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if *pci* has been populated
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *pci* is NULL
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.18.2.59 nvmlReturn_t DECLDIR nvmlDeviceGetPerformanceState (nvmlDevice_t device, nvmlPstates_t * pState)

Retrieves the current performance state for the device.

For Fermi TM or newer fully supported devices.

See [nvmlPstates_t](#) for details on allowed performance states.

Parameters:

- device* The identifier of the target device
- pState* Reference in which to return the performance state reading

Returns:

- [NVML_SUCCESS](#) if *pState* has been set

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

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

Retrieves the persistence mode associated with this device.

For all products. For Linux only.

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

See [nvmlEnableState_t](#) for details on allowed modes.

Parameters:

device The identifier of the target device

mode Reference in which to return the current driver persistence mode

Returns:

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

See also:

[nvmlDeviceSetPersistenceMode\(\)](#)

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

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

For Kepler TM or newer fully supported devices.

Parameters:

device The identifier of the target device

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

Returns:

- [NVML_SUCCESS](#) if *defaultLimit* has been set

- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *defaultLimit* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device does not support this feature
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Retrieves the power management limit associated with this device.

For Fermi TM or newer fully supported devices.

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

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

Parameters:

device The identifier of the target device

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

Returns:

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

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

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

For Kepler TM or newer fully supported devices.

Parameters:

device The identifier of the target device

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

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

Returns:

- [NVML_SUCCESS](#) if *minLimit* and *maxLimit* have been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *minLimit* or *maxLimit* is NULL

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

See also:

[nvmlDeviceSetPowerManagementLimit](#)

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

This API has been deprecated.

Retrieves the power management mode associated with this device.

For products from the Fermi family.

- Requires *NVML_INFOROM_POWER* version 3.0 or higher.

For from the Kepler or newer families.

- Does not require *NVML_INFOROM_POWER* object.

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

See [nvmlEnableState_t](#) for details on allowed modes.

Parameters:

device The identifier of the target device

mode Reference in which to return the current power management mode

Returns:

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

6.18.2.65 `nvmlReturn_t DECLDIR nvmlDeviceGetPowerState (nvmlDevice_t device, nvmlPstates_t * pState)`

Deprecated: Use [nvmlDeviceGetPerformanceState](#). This function exposes an incorrect generalization.

Retrieve the current performance state for the device.

For Fermi TM or newer fully supported devices.

See [nvmlPstates_t](#) for details on allowed performance states.

Parameters:

- device* The identifier of the target device
- pState* Reference in which to return the performance state reading

Returns:

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

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

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

For Fermi TM or newer fully supported devices.

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

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

Parameters:

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

Returns:

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

6.18.2.67 nvmlReturn_t DECLDIR nvmlDeviceGetRemappedRows (nvmlDevice_t device, unsigned int * corrRows, unsigned int * uncRows, unsigned int * isPending, unsigned int * failureOccurred)

Get number of remapped rows. The number of rows reported will be based on the cause of the remapping. *isPending* indicates whether or not there are pending remappings. A reset will be required to actually remap the row. *failureOccurred* will be set if a row remapping ever failed in the past. A pending remapping won't affect future work on the GPU since error-containment and dynamic page blacklisting will take care of that.

Note:

On MIG-enabled GPUs with active instances, querying the number of remapped rows is not supported

For newer than Volta TM fully supported devices.

Parameters:

device The identifier of the target device

corrRows Reference for number of rows remapped due to correctable errors

uncRows Reference for number of rows remapped due to uncorrectable errors

isPending Reference for whether or not remappings are pending

failureOccurred Reference that is set when a remapping has failed in the past

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_INVALID_ARGUMENT](#) If *corrRows*, *uncRows*, *isPending* or *failureOccurred* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If MIG is enabled or if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) Unexpected error

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

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

For Kepler™ or newer fully supported devices.

Parameters:

device The identifier of the target device

cause Filter page addresses by cause of retirement

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

addresses Buffer to write the page addresses into

Returns:

- [NVML_SUCCESS](#) if *pageCount* was populated and *addresses* was filled
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *pageCount* indicates the buffer is not large enough to store all the matching page addresses. *pageCount* is set to the needed size.
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid, *pageCount* is NULL, *cause* is invalid, or *addresses* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

Note:

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

For Kepler™ or newer fully supported devices.

Parameters:

device The identifier of the target device

cause Filter page addresses by cause of retirement

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

addresses Buffer to write the page addresses into

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

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

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

For Kepler™ or newer fully supported devices.

Parameters:

device The identifier of the target device

isPending Reference in which to return the pending status

Returns:

- [NVML_SUCCESS](#) if *isPending* was populated

- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *isPending* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.18.2.71 `nvmlReturn_t DECLDIR nvmlDeviceGetRowRemapperHistogram (nvmlDevice_t device, nvmlRowRemapperHistogramValues_t * values)`

Get the row remapper histogram. Returns the remap availability for each bank on the GPU.

Parameters:

device Device handle

values Histogram values

Returns:

- [NVML_SUCCESS](#) On success
- [NVML_ERROR_UNKNOWN](#) On any unexpected error

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

Gets recent samples for the GPU.

For Kepler™ or newer fully supported devices.

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

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

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

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

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

Note:

On MIG-enabled GPUs, querying the following sample types, NVML_GPU_UTILIZATION_SAMPLES, NVML_MEMORY_UTILIZATION_SAMPLES NVML_ENC_UTILIZATION_SAMPLES and NVML_DEC_UTILIZATION_SAMPLES, is not currently supported.

Parameters:

device The identifier for the target device

type Type of sampling event

lastSeenTimeStamp Return only samples with timestamp greater than lastSeenTimeStamp.

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

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

samples Reference in which samples are returned

Returns:

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

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

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

For all products with an inforom.

The serial number is an alphanumeric string that will not exceed 30 characters (including the NULL terminator). This number matches the serial number tag that is physically attached to the board. See [nvmlConstants::NVML_DEVICE_SERIAL_BUFFER_SIZE](#).

Parameters:

device The identifier of the target device

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

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

Returns:

- [NVML_SUCCESS](#) if *serial* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid, or *serial* is NULL
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *length* is too small
- [NVML_ERROR_NOT_SUPPORTED](#) if the device does not support this feature
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

For all fully supported products.

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

Parameters:

device The identifier of the target device

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

Returns:

- [NVML_SUCCESS](#) if *supportedClocksThrottleReasons* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *supportedClocksThrottleReasons* is NULL
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[NvmlClocksThrottleReasons](#)

[nvmlDeviceGetCurrentClocksThrottleReasons](#)

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

Retrieves the list of possible graphics clocks that can be used as an argument for [nvmlDeviceSetApplicationsClocks](#).

For Kepler™ or newer fully supported devices.

Parameters:

device The identifier of the target device

memoryClockMHz Memory clock for which to return possible graphics clocks

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

clocksMHz Reference in which to return the clocks in MHz

Returns:

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

See also:

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

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

Retrieves the list of possible memory clocks that can be used as an argument for [nvmlDeviceSetApplicationsClocks](#).
 For Kepler TM or newer fully supported devices.

Parameters:

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

Returns:

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

See also:

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

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

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

See [nvmlTemperatureSensors_t](#) for details on available temperature sensors.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if *temp* has been set

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

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

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

For Kepler TM or newer fully supported devices.

See [nvmlTemperatureThresholds_t](#) for details on available temperature thresholds.

Parameters:

device The identifier of the target device

thresholdType The type of threshold value queried

temp Reference in which to return the temperature reading

Returns:

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

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

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

Parameters:

device1 The identifier of the first device

device2 The identifier of the second device

pathInfo A [nvmlGpuTopologyLevel_t](#) that gives the path type

Returns:

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

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

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

Parameters:

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

Returns:

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

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

Retrieves the total ECC error counts for the device.

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

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

See [nvmlMemoryErrorType_t](#) for a description of available error types.

See [nvmlEccCounterType_t](#) for a description of available counter types.

Parameters:

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

Returns:

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

- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceClearEccErrorCounts\(\)](#)

6.18.2.82 `nvmlReturn_t DECLDIR nvmlDeviceGetTotalEnergyConsumption (nvmlDevice_t device, unsigned long long * energy)`

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

For Volta TM or newer fully supported devices.

Parameters:

device The identifier of the target device

energy Reference in which to return the energy consumption information

Returns:

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

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

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

For Fermi TM or newer fully supported devices.

See [nvmlUtilization_t](#) for details on available utilization rates.

Note:

During driver initialization when ECC is enabled one can see high GPU and Memory Utilization readings. This is caused by ECC Memory Scrubbing mechanism that is performed during driver initialization. On MIG-enabled GPUs, querying device utilization rates is not currently supported.

Parameters:

device The identifier of the target device

utilization Reference in which to return the utilization information

Returns:

- [NVML_SUCCESS](#) if *utilization* has been populated
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized

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

6.18.2.84 `nvmlReturn_t DECLDIR nvmlDeviceGetUUID (nvmlDevice_t device, char * uuid, unsigned int length)`

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

For all products.

The UUID is a globally unique identifier. It is the only available identifier for pre-Fermi-architecture products. It does NOT correspond to any identifier printed on the board. It will not exceed 96 characters in length (including the NULL terminator). See [nvmlConstants::NVML_DEVICE_UUID_V2_BUFFER_SIZE](#).

When used with MIG device handles the API returns globally unique UUIDs which can be used to identify MIG devices across both GPU and MIG devices. UUIDs are immutable for the lifetime of a MIG device.

Parameters:

- device* The identifier of the target device
- uuid* Reference in which to return the GPU UUID
- length* The maximum allowed length of the string returned in *uuid*

Returns:

- [NVML_SUCCESS](#) if *uuid* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid, or *uuid* is NULL
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *length* is too small
- [NVML_ERROR_NOT_SUPPORTED](#) if the device does not support this feature
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Get VBIOS version of the device.

For all products.

The VBIOS version may change from time to time. It will not exceed 32 characters in length (including the NULL terminator). See [nvmlConstants::NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE](#).

Parameters:

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

Returns:

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

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

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

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

Violation for thermal capping is not supported at this time.

For Kepler TM or newer fully supported devices.

Parameters:

device The identifier of the target device

perfPolicyType Represents Performance policy which can trigger GPU throttling

violTime Reference to which violation time related information is returned

Returns:

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

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

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

For all fully supported products.

Parameters:

device1 The first GPU device

device2 The second GPU device

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

Returns:

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

6.18.2.88 nvmlReturn_t DECLDIR nvmlDeviceResetApplicationsClocks (nvmlDevice_t device)

Resets the application clock to the default value

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

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

See also:

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

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

Parameters:

device The identifier of the target device

Returns:

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

6.18.2.89 nvmlReturn_t DECLDIR nvmlDeviceSetAutoBoostedClocksEnabled (nvmlDevice_t device, nvmlEnableState_t enabled)

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

For Kepler TM or newer fully supported devices.

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

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

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

Parameters:

device The identifier of the target device

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

Returns:

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

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

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

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

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

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

Parameters:

device The identifier of the target device

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

flags Flags that change the default behavior. Currently Unused.

Returns:

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

6.18.2.91 nvmlReturn_t DECLDIR nvmlDeviceValidateInforom (nvmlDevice_t device)

Reads the infoROM from the flash and verifies the checksums.

For all products with an inforom.

Parameters:

device The identifier of the target device

Returns:

- [NVML_SUCCESS](#) if infoROM is not corrupted
- [NVML_ERROR_CORRUPTED_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

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

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

Parameters:

cpuNumber The CPU number

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

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

Returns:

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

6.18.2.93 nvmlReturn_t DECLDIR nvmlVgpuInstanceGetMdevUUID (nvmlVgpuInstance_t vgpuInstance, char * mdevUuid, unsigned int size)

Retrieve the MDEV UUID of a vGPU instance.

The MDEV UUID is a globally unique identifier of the mdev device assigned to the VM, and is returned as a 5-part hexadecimal string, not exceeding 80 characters in length (including the NULL terminator). MDEV UUID is displayed only on KVM platform. See [nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE](#).

For Maxwell TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance
mdevUuid Pointer to caller-supplied buffer to hold MDEV UUID
size Size of buffer in bytes

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_NOT_SUPPORTED](#) on any hypervisor other than KVM
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *mdevUuid* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *size* is too small
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.19 CPU and Memory Affinity

Defines

- `#define NVML_AFFINITY_SCOPE_NODE 0`
Scope of NUMA node for affinity queries.
- `#define NVML_AFFINITY_SCOPE_SOCKET 1`
Scope of processor socket for affinity queries.

Functions

- `nvmlReturn_t DECLDIR nvmlDeviceGetMemoryAffinity` (`nvmlDevice_t device`, unsigned int `nodeSetSize`, unsigned long `*nodeSet`, `nvmlAffinityScope_t scope`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetCpuAffinityWithinScope` (`nvmlDevice_t device`, unsigned int `cpuSetSize`, unsigned long `*cpuSet`, `nvmlAffinityScope_t scope`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetCpuAffinity` (`nvmlDevice_t device`, unsigned int `cpuSetSize`, unsigned long `*cpuSet`)
- `nvmlReturn_t DECLDIR nvmlDeviceSetCpuAffinity` (`nvmlDevice_t device`)
- `nvmlReturn_t DECLDIR nvmlDeviceClearCpuAffinity` (`nvmlDevice_t device`)

6.19.1 Detailed Description

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

6.19.2 Function Documentation

6.19.2.1 `nvmlReturn_t DECLDIR nvmlDeviceClearCpuAffinity` (`nvmlDevice_t device`)

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

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

Parameters:

device The identifier of the target device

Returns:

- `NVML_SUCCESS` if the calling process has been successfully unbound
- `NVML_ERROR_INVALID_ARGUMENT` if *device* is invalid
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_UNKNOWN` on any unexpected error

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

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

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

Parameters:

device The identifier of the target device

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

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

Returns:

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

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

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

Note:

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

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

Parameters:

device The identifier of the target device

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

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

scope Scope that change the default behavior

Returns:

- `NVML_SUCCESS` if *cpuAffinity* has been filled
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized

- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid, *cpuSetSize* == 0, *cpuSet* is NULL or *scope* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device does not support this feature
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Retrieves an array of unsigned ints (sized to *nodeSetSize*) of bitmasks with the ideal memory affinity within node or socket for the device. For example, if NUMA node 0, 1 are ideal within the socket for the device and *nodeSetSize* == 1, *result*[0] = 0x3

Note:

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

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

Parameters:

device The identifier of the target device

nodeSetSize The size of the *nodeSet* array that is safe to access

nodeSet Array reference in which to return a bitmask of NODEs, 64 NODEs per unsigned long on 64-bit machines, 32 on 32-bit machines

scope Scope that change the default behavior

Returns:

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

6.19.2.5 `nvmlReturn_t DECLDIR nvmlDeviceSetCpuAffinity (nvmlDevice_t device)`

Sets the ideal affinity for the calling thread and device using the guidelines given in [nvmlDeviceGetCpuAffinity\(\)](#). Note, this is a change as of version 8.0. Older versions set the affinity for a calling process and all children. Currently supports up to 1024 processors.

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

Parameters:

device The identifier of the target device

Returns:

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

6.20 Unit Commands

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlUnitSetLedState](#) (nvmlUnit_t unit, [nvmlLedColor_t](#) color)

6.20.1 Detailed Description

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

6.20.2 Function Documentation

6.20.2.1 [nvmlReturn_t](#) DECLDIR [nvmlUnitSetLedState](#) (nvmlUnit_t *unit*, [nvmlLedColor_t](#) *color*)

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

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

This operation takes effect immediately.

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

See [nvmlLedColor_t](#) for available colors.

Parameters:

- unit* The identifier of the target unit
- color* The target LED color

Returns:

- [NVML_SUCCESS](#) if the LED color has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *unit* or *color* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if this is not an S-class product
- [NVML_ERROR_NO_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlUnitGetLedState\(\)](#)

6.21 Device Commands

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetPersistenceMode](#) (nvmlDevice_t device, [nvmlEnableState_t](#) mode)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetComputeMode](#) (nvmlDevice_t device, [nvmlComputeMode_t](#) mode)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetEccMode](#) (nvmlDevice_t device, [nvmlEnableState_t](#) ecc)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceClearEccErrorCounts](#) (nvmlDevice_t device, [nvmlEccCounterType_t](#) counterType)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetDriverModel](#) (nvmlDevice_t device, [nvmlDriverModel_t](#) driverModel, unsigned int flags)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetGpuLockedClocks](#) (nvmlDevice_t device, unsigned int minGpuClockMHz, unsigned int maxGpuClockMHz)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceResetGpuLockedClocks](#) (nvmlDevice_t device)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetApplicationsClocks](#) (nvmlDevice_t device, unsigned int memClockMHz, unsigned int graphicsClockMHz)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetPowerManagementLimit](#) (nvmlDevice_t device, unsigned int limit)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetGpuOperationMode](#) (nvmlDevice_t device, [nvmlGpuOperationMode_t](#) mode)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceSetAPIRestriction](#) (nvmlDevice_t device, [nvmlRestrictedAPI_t](#) apiType, [nvmlEnableState_t](#) isRestricted)

6.21.1 Detailed Description

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

6.21.2 Function Documentation

6.21.2.1 [nvmlReturn_t](#) DECLDIR [nvmlDeviceClearEccErrorCounts](#) (nvmlDevice_t *device*, [nvmlEccCounterType_t](#) *counterType*)

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

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

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

This operation takes effect immediately.

See [nvmlMemoryErrorType_t](#) for details on available counter types.

Parameters:

device The identifier of the target device

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

Returns:

- [NVML_SUCCESS](#) if the error counts were cleared
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized

- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *counterType* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device does not support this feature
- [NVML_ERROR_NO_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

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

6.21.2.2 `nvmlReturn_t DECLDIR nvmlDeviceResetGpuLockedClocks (nvmlDevice_t device)`

Resets the gpu clock to the default value

This is the gpu clock that will be used after system reboot or driver reload. Default values are idle clocks, but the current values can be changed using [nvmlDeviceSetApplicationsClocks](#).

See also:

[nvmlDeviceSetGpuLockedClocks](#)

For Volta TM or newer fully supported devices.

Parameters:

device The identifier of the target device

Returns:

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

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

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

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

Parameters:

device The identifier of the target device

apiType Target API type for this operation

isRestricted The target restriction

Returns:

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

See also:

[nvmlRestrictedAPI_t](#)

6.21.2.4 `nvmlReturn_t DECLDIR nvmlDeviceSetApplicationsClocks (nvmlDevice_t device, unsigned int memClockMHz, unsigned int graphicsClockMHz)`

Set clocks that applications will lock to.

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

Can be used as a setting to request constant performance.

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

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

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

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

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

Parameters:

device The identifier of the target device

memClockMHz Requested memory clock in MHz

graphicsClockMHz Requested graphics clock in MHz

Returns:

- [NVML_SUCCESS](#) if new settings were successfully set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized

- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *memClockMHz* and *graphicsClockMHz* is not a valid clock combination
- [NVML_ERROR_NO_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.21.2.5 `nvmlReturn_t DECLDIR nvmlDeviceSetComputeMode (nvmlDevice_t device, nvmlComputeMode_t mode)`

Set the compute mode for the device.

For all products. Requires root/admin permissions.

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

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

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

Note:

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

See [nvmlComputeMode_t](#) for details on available compute modes.

Parameters:

device The identifier of the target device

mode The target compute mode

Returns:

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

See also:

[nvmlDeviceGetComputeMode\(\)](#)

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

Set the driver model for the device.

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

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

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

This operation takes effect after the next reboot.

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

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

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

Parameters:

device The identifier of the target device

driverModel The target driver model

flags Flags that change the default behavior

Returns:

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

See also:

[nvmlDeviceGetDriverModel\(\)](#)

6.21.2.7 `nvmlReturn_t DECLDIR nvmlDeviceSetEccMode (nvmlDevice_t device, nvmlEnableState_t ecc)`

Set the ECC mode for the device.

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

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

This operation takes effect after the next reboot.

See [nvmlEnableState_t](#) for details on available modes.

Parameters:

device The identifier of the target device

ecc The target ECC mode

Returns:

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

See also:

[nvmlDeviceGetEccMode\(\)](#)

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

Set clocks that device will lock to.

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

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

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

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

Requires root/admin permissions.

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

For Volta TM or newer fully supported devices.

Parameters:

device The identifier of the target device

minGpuClockMHz Requested minimum gpu clock in MHz

maxGpuClockMHz Requested maximum gpu clock in MHz

Returns:

- [NVML_SUCCESS](#) if new settings were successfully set

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

6.21.2.9 `nvmlReturn_t DECLDIR nvmlDeviceSetGpuOperationMode (nvmlDevice_t device, nvmlGpuOperationMode_t mode)`

Sets new GOM. See [nvmlGpuOperationMode_t](#) for details.

For GK110 M-class and X-class Tesla™ products from the Kepler family. 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](#).

Parameters:

device The identifier of the target device

mode Target GOM

Returns:

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

See also:

[nvmlGpuOperationMode_t](#)
[nvmlDeviceGetGpuOperationMode](#)

6.21.2.10 `nvmlReturn_t DECLDIR nvmlDeviceSetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t mode)`

Set the persistence mode for the device.

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

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

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

See [nvmlEnableState_t](#) for available modes.

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.

Parameters:

device The identifier of the target device

mode The target persistence mode

Returns:

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

See also:

[nvmlDeviceGetPersistenceMode\(\)](#)

6.21.2.11 [nvmlReturn_t DECLDIR nvmlDeviceSetPowerManagementLimit \(nvmlDevice_t device, unsigned int limit\)](#)

Set new power limit of this device.

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

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

Note:

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

Parameters:

device The identifier of the target device

limit Power management limit in milliwatts to set

Returns:

- [NVML_SUCCESS](#) if *limit* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *defaultLimit* is out of range
- [NVML_ERROR_NOT_SUPPORTED](#) if the device does not support this feature

- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceGetPowerManagementLimitConstraints](#)

[nvmlDeviceGetPowerManagementDefaultLimit](#)

6.22 NvLink Methods

Functions

- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkState` (`nvmlDevice_t device`, `unsigned int link`, `nvmlEnableState_t *isActive`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkVersion` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int *version`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkCapability` (`nvmlDevice_t device`, `unsigned int link`, `nvmlNvLinkCapability_t capability`, `unsigned int *capResult`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkRemotePciInfo_v2` (`nvmlDevice_t device`, `unsigned int link`, `nvmlPciInfo_t *pci`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkErrorCounter` (`nvmlDevice_t device`, `unsigned int link`, `nvmlNvLinkErrorCounter_t counter`, `unsigned long long *counterValue`)
- `nvmlReturn_t DECLDIR nvmlDeviceResetNvLinkErrorCounters` (`nvmlDevice_t device`, `unsigned int link`)
- `nvmlReturn_t DECLDIR nvmlDeviceSetNvLinkUtilizationControl` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `nvmlNvLinkUtilizationControl_t *control`, `unsigned int reset`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkUtilizationControl` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `nvmlNvLinkUtilizationControl_t *control`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkUtilizationCounter` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `unsigned long long *rxcounter`, `unsigned long long *txcounter`)
- `nvmlReturn_t DECLDIR nvmlDeviceFreezeNvLinkUtilizationCounter` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`, `nvmlEnableState_t freeze`)
- `nvmlReturn_t DECLDIR nvmlDeviceResetNvLinkUtilizationCounter` (`nvmlDevice_t device`, `unsigned int link`, `unsigned int counter`)

6.22.1 Detailed Description

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

6.22.2 Function Documentation

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

Deprecated: Freezing NVLINK utilization counters is no longer supported.

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

For Pascal TM or newer fully supported devices.

Parameters:

device The identifier of the target device

link Specifies the NvLink link to be queried

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

freeze NVML_FEATURE_ENABLED = freeze the receive and transmit counters NVML_FEATURE_DISABLED = unfreeze the receive and transmit counters

Returns:

- `NVML_SUCCESS` if counters were successfully frozen or unfrozen

- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device*, *link*, *counter*, or *freeze* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

For Pascal TM or newer fully supported devices.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if *capResult* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device*, *link*, or *capability* is invalid or *capResult* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

For Pascal TM or newer fully supported devices.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if *counter* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device*, *link*, or *counter* is invalid or *counterValue* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.22.2.4 `nvmlReturn_t DECLDIR nvmlDeviceGetNvLinkRemotePciInfo_v2 (nvmlDevice_t device, unsigned int link, nvmlPciInfo_t * pci)`

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

For Pascal TM or newer fully supported devices.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if *pci* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* or *link* is invalid or *pci* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

For Pascal TM or newer fully supported devices.

Parameters:

- device* The identifier of the target device
- link* Specifies the NvLink link to be queried
- isActive* [nvmlEnableState_t](#) where [NVML_FEATURE_ENABLED](#) indicates that the link is active and [NVML_FEATURE_DISABLED](#) indicates it is inactive

Returns:

- [NVML_SUCCESS](#) if *isActive* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* or *link* is invalid or *isActive* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Deprecated: Getting utilization counter control is no longer supported.

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

For Pascal TM or newer fully supported devices.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if the control has been set successfully
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device*, *counter*, *link*, or *control* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Deprecated: Use [nvmlDeviceGetFieldValues](#) with `NVML_FI_DEV_NVLINK_THROUGHPUT_*` as field values instead.

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

For Pascal TM or newer fully supported devices.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if *rxcounter* and *txcounter* have been successfully set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device*, *counter*, or *link* is invalid or *rxcounter* or *txcounter* are NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

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

For Pascal TM or newer fully supported devices.

Parameters:

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

Returns:

- [NVML_SUCCESS](#) if *version* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* or *link* is invalid or *version* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Resets all error counters to zero Please refer to *nvmlNvLinkErrorCounter_t* for the list of error counters that are reset For Pascal TM or newer fully supported devices.

Parameters:

- device* The identifier of the target device
- link* Specifies the NVLink link to be queried

Returns:

- [NVML_SUCCESS](#) if the reset is successful
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* or *link* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Deprecated: Resetting NVLINK utilization counters is no longer supported.

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

For Pascal TM or newer fully supported devices.

Parameters:

- device* The identifier of the target device
- link* Specifies the NVLink link to be reset
- counter* Specifies the counter that should be reset (0 or 1)

Returns:

- [NVML_SUCCESS](#) if counters were successfully reset

- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device*, *link*, or *counter* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Deprecated: Setting utilization counter control is no longer supported.

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

For Pascal TM or newer fully supported devices.

Parameters:

device The identifier of the target device

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

link Specifies the NvLink link to be queried

control A reference to the [nvmlNvLinkUtilizationControl_t](#) to set

reset Resets the counters on set if non-zero

Returns:

- [NVML_SUCCESS](#) if the control has been set successfully
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device*, *counter*, *link*, or *control* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.23 Event Handling Methods

Data Structures

- struct [nvmlEventData_t](#)

Modules

- [Event Types](#)

Typedefs

- typedef struct nvmlEventSet_st * [nvmlEventSet_t](#)

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlEventSetCreate](#) ([nvmlEventSet_t](#) *set)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceRegisterEvents](#) ([nvmlDevice_t](#) device, unsigned long long eventTypes, [nvmlEventSet_t](#) set)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetSupportedEventTypes](#) ([nvmlDevice_t](#) device, unsigned long long *eventTypes)
- [nvmlReturn_t](#) DECLDIR [nvmlEventSetWait_v2](#) ([nvmlEventSet_t](#) set, [nvmlEventData_t](#) *data, unsigned int timeoutms)
- [nvmlReturn_t](#) DECLDIR [nvmlEventSetFree](#) ([nvmlEventSet_t](#) set)

6.23.1 Detailed Description

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

6.23.2 Typedef Documentation

6.23.2.1 typedef struct nvmlEventSet_st* nvmlEventSet_t

Handle to an event set

6.23.3 Function Documentation

6.23.3.1 [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetSupportedEventTypes](#) ([nvmlDevice_t](#) *device*, unsigned long long * *eventTypes*)

Returns information about events supported on device

For Fermi TM or newer fully supported devices.

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

Parameters:

device The identifier of the target device

eventTypes Reference in which to return bitmask of supported events

Returns:

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

See also:

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

6.23.3.2 `nvmlReturn_t DECLDIR nvmlDeviceRegisterEvents (nvmlDevice_t device, unsigned long long eventTypes, nvmlEventSet_t set)`

Starts recording of events on a specified devices and add the events to specified [nvmlEventSet_t](#)

For Fermi TM or newer fully supported devices. Ecc events are available only on ECC enabled devices (see [nvmlDeviceGetTotalEccErrors](#)) Power capping events are available only on Power Management enabled devices (see [nvmlDeviceGetPowerManagementMode](#))

For Linux only.

IMPORTANT: Operations on *set* are not thread safe

This call starts recording of events on specific device. All events that occurred before this call are not recorded. Checking if some event occurred can be done with [nvmlEventSetWait_v2](#)

If function reports NVML_ERROR_UNKNOWN, event set is in undefined state and should be freed. If function reports NVML_ERROR_NOT_SUPPORTED, event set can still be used. None of the requested eventTypes are registered in that case.

Parameters:

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

Returns:

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

See also:

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

6.23.3.3 `nvmlReturn_t DECLDIR nvmlEventSetCreate (nvmlEventSet_t * set)`

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

For Fermi TM or newer fully supported devices.

Parameters:

set Reference in which to return the event handle

Returns:

- [NVML_SUCCESS](#) if the event has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *set* is NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlEventSetFree](#)

6.23.3.4 `nvmlReturn_t DECLDIR nvmlEventSetFree (nvmlEventSet_t set)`

Releases events in the set

For Fermi TM or newer fully supported devices.

Parameters:

set Reference to events to be released

Returns:

- [NVML_SUCCESS](#) if the event has been successfully released
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlDeviceRegisterEvents](#)

6.23.3.5 `nvmlReturn_t DECLDIR nvmlEventSetWait_v2 (nvmlEventSet_t set, nvmlEventData_t * data, unsigned int timeoutms)`

Waits on events and delivers events

For Fermi TM or newer fully supported devices.

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

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

On Linux, every xid error event would return the associated event data and other information if applicable.

In MIG mode, if device handle is provided, the API reports all the events for the available instances, only if the caller has appropriate privileges. In absence of required privileges, only the events which affect all the instances (i.e. whole device) are reported.

This API does not currently support per-instance event reporting using MIG device handles.

Parameters:

set Reference to set of events to wait on

data Reference in which to return event data

timeoutms Maximum amount of wait time in milliseconds for registered event

Returns:

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

See also:

[Event Types](#)

[nvmlDeviceRegisterEvents](#)

6.24 Drain states

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlDeviceModifyDrainState](#) ([nvmlPciInfo_t](#) *pciInfo, [nvmlEnableState_t](#) newState)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceQueryDrainState](#) ([nvmlPciInfo_t](#) *pciInfo, [nvmlEnableState_t](#) *currentState)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceRemoveGpu_v2](#) ([nvmlPciInfo_t](#) *pciInfo, [nvmlDetachGpuState_t](#) gpuState, [nvmlPcieLinkState_t](#) linkState)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceDiscoverGpus](#) ([nvmlPciInfo_t](#) *pciInfo)

6.24.1 Detailed Description

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

6.24.2 Function Documentation

6.24.2.1 [nvmlReturn_t](#) DECLDIR [nvmlDeviceDiscoverGpus](#) ([nvmlPciInfo_t](#) * *pciInfo*)

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

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

Must be run as administrator. For Linux only.

For Pascal TM or newer fully supported devices. Some Kepler devices supported.

Parameters:

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

Returns:

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

6.24.2.2 `nvmlReturn_t DECLDIR nvmlDeviceModifyDrainState (nvmlPciInfo_t * pciInfo, nvmlEnableState_t newState)`

Modify the drain state of a GPU. This method forces a GPU to no longer accept new incoming requests. Any new NVML process will no longer see this GPU. Persistence mode for this GPU must be turned off before this call is made. Must be called as administrator. For Linux only.

For Pascal TM or newer fully supported devices. Some Kepler devices supported.

Parameters:

pciInfo The PCI address of the GPU drain state to be modified

newState The drain state that should be entered, see [nvmlEnableState_t](#)

Returns:

- [NVML_SUCCESS](#) if counters were successfully reset
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *nvmlIndex* or *newState* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_NO_PERMISSION](#) if the calling process has insufficient permissions to perform operation
- [NVML_ERROR_IN_USE](#) if the device has persistence mode turned on
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.24.2.3 `nvmlReturn_t DECLDIR nvmlDeviceQueryDrainState (nvmlPciInfo_t * pciInfo, nvmlEnableState_t * currentState)`

Query the drain state of a GPU. This method is used to check if a GPU is in a currently draining state. For Linux only.

For Pascal TM or newer fully supported devices. Some Kepler devices supported.

Parameters:

pciInfo The PCI address of the GPU drain state to be queried

currentState The current drain state for this GPU, see [nvmlEnableState_t](#)

Returns:

- [NVML_SUCCESS](#) if counters were successfully reset
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *nvmlIndex* or *currentState* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.24.2.4 `nvmlReturn_t DECLDIR nvmlDeviceRemoveGpu_v2 (nvmlPciInfo_t * pciInfo, nvmlDetachGpuState_t gpuState, nvmlPcieLinkState_t linkState)`

This method will remove the specified GPU from the view of both NVML and the NVIDIA kernel driver as long as no other processes are attached. If other processes are attached, this call will return [NVML_ERROR_IN_USE](#) and the GPU will be returned to its original "draining" state. Note: the only situation where a process can still be attached after

[nvmlDeviceModifyDrainState\(\)](#) is called to initiate the draining state is if that process was using, and is still using, a GPU before the call was made. Also note, persistence mode counts as an attachment to the GPU thus it must be disabled prior to this call.

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

For Pascal TM or newer fully supported devices. Some Kepler devices supported.

Parameters:

pciInfo The PCI address of the GPU to be removed

gpuState Whether the GPU is to be removed, from the OS see [nvmlDetachGpuState_t](#)

linkState Requested upstream PCIe link state, see [nvmlPcieLinkState_t](#)

Returns:

- [NVML_SUCCESS](#) if counters were successfully reset
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *nvmlIndex* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if the device doesn't support this feature
- [NVML_ERROR_IN_USE](#) if the device is still in use and cannot be removed

6.25 Field Value Queries

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetFieldValues](#) (nvmlDevice_t device, int valuesCount, [nvmlFieldValue_t](#) *values)

6.25.1 Detailed Description

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

6.25.2 Function Documentation

6.25.2.1 [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetFieldValues](#) (nvmlDevice_t *device*, int *valuesCount*, [nvmlFieldValue_t](#) * *values*)

Request values for a list of fields for a device. This API allows multiple fields to be queried at once. If any of the underlying fieldIds are populated by the same driver call, the results for those field IDs will be populated from a single call rather than making a driver call for each fieldId.

Parameters:

device The device handle of the GPU to request field values for

valuesCount Number of entries in values that should be retrieved

values Array of *valuesCount* structures to hold field values. Each value's fieldId must be populated prior to this call

Returns:

- [NVML_SUCCESS](#) if any values in *values* were populated. Note that you must check the nvmlReturn field of each value for each individual status
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *values* is NULL

6.26 GRID Virtualization Enums, Constants and Structs

Modules

- [GRID Virtualization Enums](#)
- [GRID Virtualization Constants](#)
- [GRID Virtualization Structs](#)

6.27 GRID Virtualization APIs

Functions

- `nvmlReturn_t` DECLDIR `nvmlDeviceGetVirtualizationMode` (`nvmlDevice_t` device, `nvmlGpuVirtualizationMode_t` *pVirtualMode)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetHostVgpuMode` (`nvmlDevice_t` device, `nvmlHostVgpuMode_t` *pHostVgpuMode)
- `nvmlReturn_t` DECLDIR `nvmlDeviceSetVirtualizationMode` (`nvmlDevice_t` device, `nvmlGpuVirtualizationMode_t` virtualMode)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetGridLicensableFeatures_v3` (`nvmlDevice_t` device, `nvmlGridLicensableFeatures_t` *pGridLicensableFeatures)
- `nvmlReturn_t` DECLDIR `nvmlDeviceGetProcessUtilization` (`nvmlDevice_t` device, `nvmlProcessUtilizationSample_t` *utilization, unsigned int *processSamplesCount, unsigned long long lastSeenTimeStamp)

6.27.1 Detailed Description

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

6.27.2 Function Documentation

6.27.2.1 `nvmlReturn_t` DECLDIR `nvmlDeviceGetGridLicensableFeatures_v3` (`nvmlDevice_t` device, `nvmlGridLicensableFeatures_t` *pGridLicensableFeatures)

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.

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

6.27.2.2 `nvmlReturn_t` DECLDIR `nvmlDeviceGetHostVgpuMode` (`nvmlDevice_t` device, `nvmlHostVgpuMode_t` *pHostVgpuMode)

Queries if SR-IOV host operation is supported on a vGPU supported device.

Checks whether SR-IOV host capability is supported by the device and the driver, and indicates device is in SR-IOV mode if both of these conditions are true.

Parameters:

device The identifier of the target device

pHostVgpuMode Reference in which to return the current vGPU mode

Returns:

- [NVML_SUCCESS](#) if device's vGPU mode has been successfully retrieved
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* handle is 0 or *pVgpuMode* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if *device* doesn't support this feature.
- [NVML_ERROR_UNKNOWN](#) if any unexpected error occurred

6.27.2.3 `nvmlReturn_t DECLDIR nvmlDeviceGetProcessUtilization (nvmlDevice_t device, nvmlProcessUtilizationSample_t * utilization, unsigned int * processSamplesCount, unsigned long long lastSeenTimeStamp)`

Retrieves the current utilization and process ID

For Maxwell TM or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder for processes running. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by *utilization*. One utilization sample structure is returned per process running, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with *utilization* set to NULL. The caller should allocate a buffer of size `processSamplesCount * sizeof(nvmlProcessUtilizationSample_t)`. Invoke the function again with the allocated buffer passed in *utilization*, and *processSamplesCount* set to the number of entries the buffer is sized for.

On successful return, the function updates *processSamplesCount* with the number of process utilization sample structures that were actually written. This may differ from a previously read value as instances are created or destroyed.

lastSeenTimeStamp represents the CPU timestamp in microseconds at which utilization samples were last read. Set it to 0 to read utilization based on all the samples maintained by the driver's internal sample buffer. Set *lastSeenTimeStamp* to a timestamp retrieved from a previous query to read utilization since the previous query.

Note:

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

Parameters:

device The identifier of the target device

utilization Pointer to caller-supplied buffer in which guest process utilization samples are returned

processSamplesCount Pointer to caller-supplied array size, and returns number of processes running

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

Returns:

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

6.27.2.4 nvmlReturn_t DECLDIR nvmlDeviceGetVirtualizationMode (nvmlDevice_t device, nvmlGpuVirtualizationMode_t * pVirtualMode)

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

For Kepler TM or newer fully supported devices.

Parameters:

device Identifier of the target device

pVirtualMode Reference to virtualization mode. One of NVML_GPU_VIRTUALIZATION_?

Returns:

- [NVML_SUCCESS](#) if *pVirtualMode* is fetched
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid or *pVirtualMode* is NULL
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.27.2.5 nvmlReturn_t DECLDIR nvmlDeviceSetVirtualizationMode (nvmlDevice_t device, nvmlGpuVirtualizationMode_t virtualMode)

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

For Kepler TM or newer fully supported devices.

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.

6.28 GRID vGPU Management

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetSupportedVgpus](#) (nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuTypeId_t *vgpuTypeIds)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetCreatableVgpus](#) (nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuTypeId_t *vgpuTypeIds)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetClass](#) (nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeClass, unsigned int *size)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetName](#) (nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeName, unsigned int *size)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetDeviceID](#) (nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *deviceID, unsigned long long *subsystemID)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetFramebufferSize](#) (nvmlVgpuTypeId_t vgpuTypeId, unsigned long long *fbSize)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetNumDisplayHeads](#) (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *numDisplayHeads)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetResolution](#) (nvmlVgpuTypeId_t vgpuTypeId, unsigned int displayIndex, unsigned int *xdim, unsigned int *ydim)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetLicense](#) (nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeLicenseString, unsigned int size)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetFrameRateLimit](#) (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *frameRateLimit)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetMaxInstances](#) (nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCount)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuTypeGetMaxInstancesPerVm](#) (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *vgpuInstanceCountPerVm)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetActiveVgpus](#) (nvmlDevice_t device, unsigned int *vgpuCount, nvmlVgpuInstance_t *vgpuInstances)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetVmID](#) (nvmlVgpuInstance_t vgpuInstance, char *vmId, unsigned int size, [nvmlVgpuVmIdType_t](#) *vmIdType)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetUUID](#) (nvmlVgpuInstance_t vgpuInstance, char *uuid, unsigned int size)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetVmDriverVersion](#) (nvmlVgpuInstance_t vgpuInstance, char *version, unsigned int length)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetFbUsage](#) (nvmlVgpuInstance_t vgpuInstance, unsigned long long *fbUsage)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetLicenseStatus](#) (nvmlVgpuInstance_t vgpuInstance, unsigned int *licensed)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetType](#) (nvmlVgpuInstance_t vgpuInstance, nvmlVgpuTypeId_t *vgpuTypeId)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetFrameRateLimit](#) (nvmlVgpuInstance_t vgpuInstance, unsigned int *frameRateLimit)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetEccMode](#) (nvmlVgpuInstance_t vgpuInstance, [nvmlEnableState_t](#) *eccMode)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetEncoderCapacity](#) (nvmlVgpuInstance_t vgpuInstance, unsigned int *encoderCapacity)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceSetEncoderCapacity](#) (nvmlVgpuInstance_t vgpuInstance, unsigned int encoderCapacity)
- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetEncoderStats](#) (nvmlVgpuInstance_t vgpuInstance, unsigned int *sessionCount, unsigned int *averageFps, unsigned int *averageLatency)

- `nvmlReturn_t` DECLDIR `nvmlVgpuInstanceGetEncoderSessions` (`nvmlVgpuInstance_t` *vgpuInstance*, unsigned int **sessionCount*, `nvmlEncoderSessionInfo_t` **sessionInfo*)
- `nvmlReturn_t` DECLDIR `nvmlVgpuInstanceGetFBCStats` (`nvmlVgpuInstance_t` *vgpuInstance*, `nvmlFBCStats_t` **fbcStats*)
- `nvmlReturn_t` DECLDIR `nvmlVgpuInstanceGetFBCSessions` (`nvmlVgpuInstance_t` *vgpuInstance*, unsigned int **sessionCount*, `nvmlFBCSessionInfo_t` **sessionInfo*)

6.28.1 Detailed Description

This chapter describes APIs supporting NVIDIA GRID vGPU.

6.28.2 Function Documentation

6.28.2.1 `nvmlReturn_t` DECLDIR `nvmlDeviceGetActiveVgpus` (`nvmlDevice_t` *device*, unsigned int **vgpuCount*, `nvmlVgpuInstance_t` **vgpuInstances*)

Retrieve the active vGPU instances on a device.

An array of active vGPU instances is returned in the caller-supplied buffer pointed at by *vgpuInstances*. The array element count is passed in *vgpuCount*, and *vgpuCount* is used to return the number of vGPU instances written to the buffer.

If the supplied buffer is not large enough to accomodate the vGPU instance array, the function returns `NVML_ERROR_INSUFFICIENT_SIZE`, with the element count of `nvmlVgpuInstance_t` array required in *vgpuCount*. To query the number of active vGPU instances, call this function with **vgpuCount* = 0. The code will return `NVML_ERROR_INSUFFICIENT_SIZE`, or `NVML_SUCCESS` if no vGPU Types are supported.

For Kepler TM or newer fully supported devices.

Parameters:

device The identifier of the target device

vgpuCount Pointer which passes in the array size as well as get back the number of types

vgpuInstances Pointer to array in which to return list of vGPU instances

Returns:

- `NVML_SUCCESS` successful completion
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if *device* is invalid, or *vgpuCount* is NULL
- `NVML_ERROR_INSUFFICIENT_SIZE` if *size* is too small
- `NVML_ERROR_NOT_SUPPORTED` if vGPU is not supported by the device
- `NVML_ERROR_UNKNOWN` on any unexpected error

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

Retrieve the currently creatable vGPU types on a physical GPU (device).

An array of creatable vGPU types for the physical GPU indicated by *device* is returned in the caller-supplied buffer pointed at by *vgpuTypeIds*. The element count of `nvmlVgpuTypeId_t` array is passed in *vgpuCount*, and *vgpuCount* is used to return the number of vGPU types written to the buffer.

The creatable vGPU types for a device may differ over time, as there may be restrictions on what type of vGPU types can concurrently run on a device. For example, if only one vGPU type is allowed at a time on a device, then the creatable list will be restricted to whatever vGPU type is already running on the device.

If the supplied buffer is not large enough to accommodate the vGPU type array, the function returns `NVML_ERROR_INSUFFICIENT_SIZE`, with the element count of `nvmlVgpuTypeId_t` array required in *vgpuCount*. To query the number of vGPU types creatable for the GPU, call this function with `*vgpuCount = 0`. The code will return `NVML_ERROR_INSUFFICIENT_SIZE`, or `NVML_SUCCESS` if no vGPU types are creatable.

Parameters:

- device* The identifier of the target device
- vgpuCount* Pointer to caller-supplied array size, and returns number of vGPU types
- vgpuTypeIds* Pointer to caller-supplied array in which to return list of vGPU types

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_INSUFFICIENT_SIZE](#) *vgpuTypeIds* buffer is too small, array element count is returned in *vgpuCount*
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuCount* is NULL
- [NVML_ERROR_NOT_SUPPORTED](#) if vGPU is not supported by the device
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.3 `nvmlReturn_t DECLDIR nvmlDeviceGetSupportedVgpus (nvmlDevice_t device, unsigned int * vgpuCount, nvmlVgpuTypeId_t * vgpuTypeIds)`

Retrieve the supported vGPU types on a physical GPU (device).

An array of supported vGPU types for the physical GPU indicated by *device* is returned in the caller-supplied buffer pointed at by *vgpuTypeIds*. The element count of `nvmlVgpuTypeId_t` array is passed in *vgpuCount*, and *vgpuCount* is used to return the number of vGPU types written to the buffer.

If the supplied buffer is not large enough to accommodate the vGPU type array, the function returns `NVML_ERROR_INSUFFICIENT_SIZE`, with the element count of `nvmlVgpuTypeId_t` array required in *vgpuCount*. To query the number of vGPU types supported for the GPU, call this function with `*vgpuCount = 0`. The code will return `NVML_ERROR_INSUFFICIENT_SIZE`, or `NVML_SUCCESS` if no vGPU types are supported.

Parameters:

- device* The identifier of the target device
- vgpuCount* Pointer to caller-supplied array size, and returns number of vGPU types
- vgpuTypeIds* Pointer to caller-supplied array in which to return list of vGPU types

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_INSUFFICIENT_SIZE](#) *vgpuTypeIds* buffer is too small, array element count is returned in *vgpuCount*
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuCount* is NULL or *device* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if vGPU is not supported by the device
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.4 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetEccMode (nvmlVgpuInstance_t vgpuInstance, nvmlEnableState_t * eccMode)`

Retrieve the current ECC mode of vGPU instance.

Parameters:

vgpuInstance The identifier of the target vGPU instance
eccMode Reference in which to return the current ECC mode

Returns:

- [NVML_SUCCESS](#) if the *vgpuInstance*'s ECC mode has been successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *mode* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_NOT_SUPPORTED](#) if the vGPU doesn't support this feature
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.5 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetEncoderCapacity (nvmlVgpuInstance_t vgpuInstance, unsigned int * encoderCapacity)`

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

For Maxwell TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance
encoderCapacity Reference to an unsigned int for the encoder capacity

Returns:

- [NVML_SUCCESS](#) if *encoderCapacity* has been retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *encoderQueryType* is invalid
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.6 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetEncoderSessions (nvmlVgpuInstance_t vgpuInstance, unsigned int * sessionCount, nvmlEncoderSessionInfo_t * sessionInfo)`

Retrieves information about all active encoder sessions on a vGPU Instance.

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

If the supplied buffer is not large enough to accommodate the active session array, the function returns [NVML_ERROR_INSUFFICIENT_SIZE](#), with the element count of [nvmlEncoderSessionInfo_t](#) array required in *sessionCount*. To

query the number of active encoder sessions, call this function with `*sessionCount = 0`. The code will return `NVML_SUCCESS` with number of active encoder sessions updated in `*sessionCount`.

For Maxwell TM or newer fully supported devices.

Parameters:

- vgpuInstance*** Identifier of the target vGPU instance
- sessionCount*** Reference to caller supplied array size, and returns the number of sessions.
- sessionInfo*** Reference to caller supplied array in which the list of session information is 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

6.28.2.7 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetEncoderStats (nvmlVgpuInstance_t vgpuInstance, unsigned int * sessionCount, unsigned int * averageFps, unsigned int * averageLatency)`

Retrieves the current encoder statistics of a vGPU Instance

For Maxwell TM or newer fully supported devices.

Parameters:

- vgpuInstance*** Identifier of the target vGPU instance
- sessionCount*** Reference to an unsigned int for count of active encoder sessions
- averageFps*** Reference to an unsigned int for trailing average FPS of all active sessions
- averageLatency*** Reference to an unsigned int for encode latency in microseconds

Returns:

- `NVML_SUCCESS` if *sessionCount*, *averageFps* and *averageLatency* is fetched
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if *sessionCount*, or *averageFps* or *averageLatency* is NULL or *vgpuInstance* is 0.
- `NVML_ERROR_NOT_FOUND` if *vgpuInstance* does not match a valid active vGPU instance on the system
- `NVML_ERROR_UNKNOWN` on any unexpected error

6.28.2.8 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetFBCSessions (nvmlVgpuInstance_t vgpuInstance, unsigned int * sessionCount, nvmlFBCSessionInfo_t * sessionInfo)`

Retrieves information about active frame buffer capture sessions on a vGPU Instance.

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

If the supplied buffer is not large enough to accomodate the active session array, the function returns NVML_ERROR_INSUFFICIENT_SIZE, with the element count of [nvmlFBCSessionInfo_t](#) array required in *sessionCount*. To query the number of active FBC sessions, call this function with *sessionCount = 0. The code will return NVML_SUCCESS with number of active FBC sessions updated in *sessionCount.

For Maxwell TM or newer fully supported devices.

Note:

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

Parameters:

vgpuInstance Identifier of the target vGPU instance

sessionCount Reference to caller supplied array size, and returns the number of sessions.

sessionInfo Reference in which to return the session information

Returns:

- NVML_SUCCESS if *sessionInfo* is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if *vgpuInstance* is 0, or *sessionCount* is NULL.
- NVML_ERROR_NOT_FOUND if *vgpuInstance* does not match a valid active vGPU instance on the system
- NVML_ERROR_INSUFFICIENT_SIZE if *sessionCount* is too small, array element count is returned in *sessionCount*
- NVML_ERROR_UNKNOWN on any unexpected error

6.28.2.9 nvmlReturn_t DECLDIR nvmlVgpuInstanceGetFBCStats (nvmlVgpuInstance_t vgpuInstance, nvmlFBCStats_t * fbcStats)

Retrieves the active frame buffer capture sessions statistics of a vGPU Instance

For Maxwell TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance

fbcStats Reference to [nvmlFBCStats_t](#) structure containing NvFBC stats

Returns:

- NVML_SUCCESS if *fbcStats* is fetched
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if *vgpuInstance* is 0, or *fbcStats* is NULL
- NVML_ERROR_NOT_FOUND if *vgpuInstance* does not match a valid active vGPU instance on the system
- NVML_ERROR_UNKNOWN on any unexpected error

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

Retrieve the framebuffer usage in bytes.

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

For Kepler TM or newer fully supported devices.

Parameters:

vgpuInstance The identifier of the target instance

fbUsage Pointer to framebuffer usage in bytes

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *fbUsage* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.11 **nvmlReturn_t DECLDIR nvmlVgpuInstanceGetFrameRateLimit (nvmlVgpuInstance_t *vgpuInstance*, unsigned int **frameRateLimit*)**

Retrieve the frame rate limit set for the vGPU instance.

Returns the value of the frame rate limit set for the vGPU instance

For Kepler TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance

frameRateLimit Reference to return the frame rate limit

Returns:

- [NVML_SUCCESS](#) if *frameRateLimit* has been set
- [NVML_ERROR_NOT_SUPPORTED](#) if frame rate limiter is turned off for the vGPU type
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *frameRateLimit* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.12 **nvmlReturn_t DECLDIR nvmlVgpuInstanceGetLicenseStatus (nvmlVgpuInstance_t *vgpuInstance*, unsigned int **licensed*)**

Retrieve the current licensing state of the vGPU instance.

If the vGPU is currently licensed, *licensed* is set to 1, otherwise it is set to 0.

For Kepler TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance

licensed Reference to return the licensing status

Returns:

- [NVML_SUCCESS](#) if *licensed* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *licensed* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.13 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetType (nvmlVgpuInstance_t vgpuInstance, nvmlVgpuTypeId_t * vgpuTypeId)`

Retrieve the vGPU type of a vGPU instance.

Returns the vGPU type ID of vgpu assigned to the vGPU instance.

For Kepler TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance

vgpuTypeId Reference to return the vgpuTypeId

Returns:

- [NVML_SUCCESS](#) if *vgpuTypeId* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *vgpuTypeId* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.14 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetUUID (nvmlVgpuInstance_t vgpuInstance, char * uuid, unsigned int size)`

Retrieve the UUID of a vGPU instance.

The UUID is a globally unique identifier associated with the vGPU, and is returned as a 5-part hexadecimal string, not exceeding 80 characters in length (including the NULL terminator). See [nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE](#).

For Kepler TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance

uuid Pointer to caller-supplied buffer to hold vGPU UUID

size Size of buffer in bytes

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *uuid* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *size* is too small
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Retrieve the NVIDIA driver version installed in the VM associated with a vGPU.

The version is returned as an alphanumeric string in the caller-supplied buffer *version*. The length of the version string will not exceed 80 characters in length (including the NUL terminator). See [nvmlConstants::NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE](#).

`nvmlVgpuInstanceGetVmDriverVersion()` may be called at any time for a vGPU instance. The guest VM driver version is returned as "Not Available" if no NVIDIA driver is installed in the VM, or the VM has not yet booted to the point where the NVIDIA driver is loaded and initialized.

For Kepler TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance
version Caller-supplied buffer to return driver version string
length Size of *version* buffer

Returns:

- [NVML_SUCCESS](#) if *version* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *length* is too small
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Retrieve the VM ID associated with a vGPU instance.

The VM ID is returned as a string, not exceeding 80 characters in length (including the NUL terminator). See [nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE](#).

The format of the VM ID varies by platform, and is indicated by the type identifier returned in *vmIdType*.

For Kepler TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance

vmId Pointer to caller-supplied buffer to hold VM ID

size Size of buffer in bytes

vmIdType Pointer to hold VM ID type

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vmId* or *vmIdType* is NULL, or *vgpuInstance* is 0
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *size* is too small
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.17 `nvmlReturn_t DECLDIR nvmlVgpuInstanceSetEncoderCapacity (nvmlVgpuInstance_t vgpuInstance, unsigned int encoderCapacity)`

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

For Maxwell TM or newer fully supported devices.

Parameters:

vgpuInstance Identifier of the target vGPU instance

encoderCapacity Unsigned int for the encoder capacity value

Returns:

- [NVML_SUCCESS](#) if *encoderCapacity* has been set
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *encoderCapacity* is out of range of 0-100.
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Retrieve the class of a vGPU type. It will not exceed 64 characters in length (including the NUL terminator). See [nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE](#).

For Kepler TM or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type

vgpuTypeClass Pointer to string array to return class in

size Size of string

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *vgpuTypeClass* is NULL
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *size* is too small
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.19 `nvmlReturn_t DECLDIR nvmlVgpuTypeGetDeviceID (nvmlVgpuTypeId_t vgpuTypeId, unsigned long long * deviceId, unsigned long long * subsystemID)`

Retrieve the device ID of a vGPU type.

For Kepler TM or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type

deviceId Device ID and vendor ID of the device contained in single 32 bit value

subsystemID Subsystem ID and subsystem vendor ID of the device contained in single 32 bit value

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *deviceId* or *subsystemID* are NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.20 `nvmlReturn_t DECLDIR nvmlVgpuTypeGetFramebufferSize (nvmlVgpuTypeId_t vgpuTypeId, unsigned long long * fbSize)`

Retrieve the vGPU framebuffer size in bytes.

For Kepler TM or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type

fbSize Pointer to framebuffer size in bytes

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *fbSize* is NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.21 `nvmlReturn_t DECLDIR nvmlVgpuTypeGetFrameRateLimit (nvmlVgpuTypeId_t vgpuTypeId, unsigned int *frameRateLimit)`

Retrieve the static frame rate limit value of the vGPU type

For Kepler™ or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type

frameRateLimit Reference to return the frame rate limit value

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_NOT_SUPPORTED](#) if frame rate limiter is turned off for the vGPU type
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *frameRateLimit* is NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.22 `nvmlReturn_t DECLDIR nvmlVgpuTypeGetLicense (nvmlVgpuTypeId_t vgpuTypeId, char *vgpuTypeLicenseString, unsigned int size)`

Retrieve license requirements for a vGPU type

The license type and version required to run the specified vGPU type is returned as an alphanumeric string, in the form "<license name>,<version>", for example "GRID-Virtual-PC,2.0". If a vGPU is runnable with* more than one type of license, the licenses are delimited by a semicolon, for example "GRID-Virtual-PC,2.0;GRID-Virtual-WS,2.0;GRID-Virtual-WS-Ext,2.0".

The total length of the returned string will not exceed 128 characters, including the NUL terminator. See [nvmlVgpuConstants::NVML_GRID_LICENSE_BUFFER_SIZE](#).

For Kepler™ or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type

vgpuTypeLicenseString Pointer to buffer to return license info

size Size of *vgpuTypeLicenseString* buffer

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *vgpuTypeLicenseString* is NULL
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *size* is too small
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.23 `nvmlReturn_t DECLDIR nvmlVgpuTypeGetMaxInstances (nvmlDevice_t device, nvmlVgpuTypeId_t vgpuTypeId, unsigned int * vgpuInstanceCount)`

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

For Kepler TM or newer fully supported devices.

Parameters:

device The identifier of the target device

vgpuTypeId Handle to vGPU type

vgpuInstanceCount Pointer to get the max number of vGPU instances that can be created on a device for given vgpuTypeId

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid or is not supported on target device, or *vgpuInstanceCount* is NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.24 `nvmlReturn_t DECLDIR nvmlVgpuTypeGetMaxInstancesPerVm (nvmlVgpuTypeId_t vgpuTypeId, unsigned int * vgpuInstanceCountPerVm)`

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

For Kepler TM or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type

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

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *vgpuInstanceCountPerVm* is NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

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

Retrieve the vGPU type name.

The name is an alphanumeric string that denotes a particular vGPU, e.g. GRID M60-2Q. It will not exceed 64 characters in length (including the NUL terminator). See [nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE](#).

For Kepler TM or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type
vgpuTypeName Pointer to buffer to return name
size Size of buffer

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *name* is NULL
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *size* is too small
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.26 `nvmlReturn_t DECLDIR nvmlVgpuTypeGetNumDisplayHeads (nvmlVgpuTypeId_t vgpuTypeId, unsigned int * numDisplayHeads)`

Retrieve count of vGPU's supported display heads.

For Kepler™ or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type
numDisplayHeads Pointer to number of display heads

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *numDisplayHeads* is NULL
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.28.2.27 `nvmlReturn_t DECLDIR nvmlVgpuTypeGetResolution (nvmlVgpuTypeId_t vgpuTypeId, unsigned int displayIndex, unsigned int * xdim, unsigned int * ydim)`

Retrieve vGPU display head's maximum supported resolution.

For Kepler™ or newer fully supported devices.

Parameters:

vgpuTypeId Handle to vGPU type
displayIndex Zero-based index of display head
xdim Pointer to maximum number of pixels in X dimension
ydim Pointer to maximum number of pixels in Y dimension

Returns:

- [NVML_SUCCESS](#) successful completion
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuTypeId* is invalid, or *xdim* or *ydim* are NULL, or *displayIndex* is out of range.
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.29 GRID Virtualization Migration

Data Structures

- struct [nvmlVgpuVersion_t](#)
- struct [nvmlVgpuMetadata_t](#)
- struct [nvmlVgpuPgpuMetadata_t](#)
- struct [nvmlVgpuPgpuCompatibility_t](#)

Enumerations

- enum [nvmlVgpuVmCompatibility_t](#) {
[NVML_VGPU_VM_COMPATIBILITY_NONE](#) = 0x0,
[NVML_VGPU_VM_COMPATIBILITY_COLD](#) = 0x1,
[NVML_VGPU_VM_COMPATIBILITY_HIBERNATE](#) = 0x2,
[NVML_VGPU_VM_COMPATIBILITY_SLEEP](#) = 0x4,
[NVML_VGPU_VM_COMPATIBILITY_LIVE](#) = 0x8 }
- enum [nvmlVgpuPgpuCompatibilityLimitCode_t](#) {
[NVML_VGPU_COMPATIBILITY_LIMIT_NONE](#) = 0x0,
[NVML_VGPU_COMPATIBILITY_LIMIT_HOST_DRIVER](#) = 0x1,
[NVML_VGPU_COMPATIBILITY_LIMIT_GUEST_DRIVER](#) = 0x2,
[NVML_VGPU_COMPATIBILITY_LIMIT_GPU](#) = 0x4,
[NVML_VGPU_COMPATIBILITY_LIMIT_OTHER](#) = 0x80000000 }

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlVgpuInstanceGetMetadata](#) ([nvmlVgpuInstance_t](#) vgpuInstance, [nvmlVgpuMetadata_t](#) *vgpuMetadata, unsigned int *bufferSize)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetVgpuMetadata](#) ([nvmlDevice_t](#) device, [nvmlVgpuPgpuMetadata_t](#) *pgpuMetadata, unsigned int *bufferSize)
- [nvmlReturn_t](#) DECLDIR [nvmlGetVgpuCompatibility](#) ([nvmlVgpuMetadata_t](#) *vgpuMetadata, [nvmlVgpuPgpuMetadata_t](#) *pgpuMetadata, [nvmlVgpuPgpuCompatibility_t](#) *compatibilityInfo)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetPgpuMetadataString](#) ([nvmlDevice_t](#) device, char *pgpuMetadata, unsigned int *bufferSize)
- [nvmlReturn_t](#) DECLDIR [nvmlGetVgpuVersion](#) ([nvmlVgpuVersion_t](#) *supported, [nvmlVgpuVersion_t](#) *current)
- [nvmlReturn_t](#) DECLDIR [nvmlSetVgpuVersion](#) ([nvmlVgpuVersion_t](#) *vgpuVersion)

6.29.1 Detailed Description

This chapter describes operations that are associated with vGPU Migration.

6.29.2 Enumeration Type Documentation

6.29.2.1 enum nvmlVgpuPgpuCompatibilityLimitCode_t

vGPU-pGPU compatibility limit codes

Enumerator:

NVML_VGPU_COMPATIBILITY_LIMIT_NONE Compatibility is not limited.
NVML_VGPU_COMPATIBILITY_LIMIT_HOST_DRIVER ompatibility is limited by host driver version.
NVML_VGPU_COMPATIBILITY_LIMIT_GUEST_DRIVER Compatibility is limited by guest driver version.
NVML_VGPU_COMPATIBILITY_LIMIT_GPU Compatibility is limited by GPU hardware.
NVML_VGPU_COMPATIBILITY_LIMIT_OTHER Compatibility is limited by an undefined factor.

6.29.2.2 enum nvmlVgpuVmCompatibility_t

vGPU VM compatibility codes

Enumerator:

NVML_VGPU_VM_COMPATIBILITY_NONE vGPU is not runnable
NVML_VGPU_VM_COMPATIBILITY_COLD vGPU is runnable from a cold / powered-off state (ACPI S5)
NVML_VGPU_VM_COMPATIBILITY_HIBERNATE vGPU is runnable from a hibernated state (ACPI S4)
NVML_VGPU_VM_COMPATIBILITY_SLEEP vGPU is runnable from a slept state (ACPI S3)
NVML_VGPU_VM_COMPATIBILITY_LIVE vGPU is runnable from a live/paused (ACPI S0)

6.29.3 Function Documentation

6.29.3.1 nvmlReturn_t DECLDIR nvmlDeviceGetPgpuMetadataString (nvmlDevice_t device, char * pgpuMetadata, unsigned int * bufferSize)

Returns the properties of the physical GPU indicated by the device in an ascii-encoded string format.

The caller passes in a buffer via *pgpuMetadata*, with the size of the buffer in *bufferSize*. If the string is too large to fit in the supplied buffer, the function returns NVML_ERROR_INSUFFICIENT_SIZE with the size needed in *bufferSize*.

Parameters:

device The identifier of the target device
pgpuMetadata Pointer to caller-supplied buffer into which *pgpuMetadata* is written
bufferSize Pointer to size of *pgpuMetadata* buffer

Returns:

- **NVML_SUCCESS** GPU metadata structure was successfully returned
- **NVML_ERROR_INSUFFICIENT_SIZE** *pgpuMetadata* buffer is too small, required size is returned in *bufferSize*
- **NVML_ERROR_INVALID_ARGUMENT** if *bufferSize* is NULL or *device* is invalid; if *pgpuMetadata* is NULL and the value of *bufferSize* is not 0.
- **NVML_ERROR_NOT_SUPPORTED** if vGPU is not supported by the system
- **NVML_ERROR_UNKNOWN** on any unexpected error

6.29.3.2 `nvmlReturn_t DECLDIR nvmlDeviceGetVgpuMetadata (nvmlDevice_t device, nvmlVgpuPgpuMetadata_t * pgpuMetadata, unsigned int * bufferSize)`

Returns a vGPU metadata structure for the physical GPU indicated by *device*. The structure contains information about the GPU and the currently installed NVIDIA host driver version that's controlling it, together with an opaque data section containing internal state.

The caller passes in a buffer via *pgpuMetadata*, with the size of the buffer in *bufferSize*. If the *pgpuMetadata* structure is too large to fit in the supplied buffer, the function returns `NVML_ERROR_INSUFFICIENT_SIZE` with the size needed in *bufferSize*.

Parameters:

device The identifier of the target device

pgpuMetadata Pointer to caller-supplied buffer into which *pgpuMetadata* is written

bufferSize Pointer to size of *pgpuMetadata* buffer

Returns:

- `NVML_SUCCESS` GPU metadata structure was successfully returned
- `NVML_ERROR_INSUFFICIENT_SIZE` *pgpuMetadata* buffer is too small, required size is returned in *bufferSize*
- `NVML_ERROR_INVALID_ARGUMENT` if *bufferSize* is NULL or *device* is invalid; if *pgpuMetadata* is NULL and the value of *bufferSize* is not 0.
- `NVML_ERROR_NOT_SUPPORTED` vGPU is not supported by the system
- `NVML_ERROR_UNKNOWN` on any unexpected error

6.29.3.3 `nvmlReturn_t DECLDIR nvmlGetVgpuCompatibility (nvmlVgpuMetadata_t * vgpuMetadata, nvmlVgpuPgpuMetadata_t * pgpuMetadata, nvmlVgpuPgpuCompatibility_t * compatibilityInfo)`

Takes a vGPU instance metadata structure read from `nvmlVgpuInstanceGetMetadata()`, and a vGPU metadata structure for a physical GPU read from `nvmlDeviceGetVgpuMetadata()`, and returns compatibility information of the vGPU instance and the physical GPU.

The caller passes in a buffer via *compatibilityInfo*, into which a compatibility information structure is written. The structure defines the states in which the vGPU / VM may be booted on the physical GPU. If the vGPU / VM compatibility with the physical GPU is limited, a limit code indicates the factor limiting compability. (see `nvmlVgpuPgpuCompatibilityLimitCode_t` for details).

Note: vGPU compatibility does not take into account dynamic capacity conditions that may limit a system's ability to boot a given vGPU or associated VM.

Parameters:

vgpuMetadata Pointer to caller-supplied vGPU metadata structure

pgpuMetadata Pointer to caller-supplied GPU metadata structure

compatibilityInfo Pointer to caller-supplied buffer to hold compatibility info

Returns:

- `NVML_SUCCESS` vGPU metadata structure was successfully returned
- `NVML_ERROR_INVALID_ARGUMENT` if *vgpuMetadata* or *pgpuMetadata* or *bufferSize* are NULL
- `NVML_ERROR_UNKNOWN` on any unexpected error

6.29.3.4 `nvmlReturn_t DECLDIR nvmlGetVgpuVersion (nvmlVgpuVersion_t * supported, nvmlVgpuVersion_t * current)`

Query the ranges of supported vGPU versions.

This function gets the linear range of supported vGPU versions that is preset for the NVIDIA vGPU Manager and the range set by an administrator. If the preset range has not been overridden by [nvmlSetVgpuVersion](#), both ranges are the same.

The caller passes pointers to the following [nvmlVgpuVersion_t](#) structures, into which the NVIDIA vGPU Manager writes the ranges: 1. *supported* structure that represents the preset range of vGPU versions supported by the NVIDIA vGPU Manager. 2. *current* structure that represents the range of supported vGPU versions set by an administrator. By default, this range is the same as the preset range.

Parameters:

- supported* Pointer to the structure in which the preset range of vGPU versions supported by the NVIDIA vGPU Manager is written
- current* Pointer to the structure in which the range of supported vGPU versions set by an administrator is written

Returns:

- [NVML_SUCCESS](#) The vGPU version range structures were successfully obtained.
- [NVML_ERROR_NOT_SUPPORTED](#) The API is not supported.
- [NVML_ERROR_INVALID_ARGUMENT](#) The *supported* parameter or the *current* parameter is NULL.
- [NVML_ERROR_UNKNOWN](#) An error occurred while the data was being fetched.

6.29.3.5 `nvmlReturn_t DECLDIR nvmlSetVgpuVersion (nvmlVgpuVersion_t * vgpuVersion)`

Override the preset range of vGPU versions supported by the NVIDIA vGPU Manager with a range set by an administrator.

This function configures the NVIDIA vGPU Manager with a range of supported vGPU versions set by an administrator. This range must be a subset of the preset range that the NVIDIA vGPU Manager supports. The custom range set by an administrator takes precedence over the preset range and is advertised to the guest VM for negotiating the vGPU version. See [nvmlGetVgpuVersion](#) for details of how to query the preset range of versions supported.

This function takes a pointer to vGPU version range structure [nvmlVgpuVersion_t](#) as input to override the preset vGPU version range that the NVIDIA vGPU Manager supports.

After host system reboot or driver reload, the range of supported versions reverts to the range that is preset for the NVIDIA vGPU Manager.

Note:

1. The range set by the administrator must be a subset of the preset range that the NVIDIA vGPU Manager supports. Otherwise, an error is returned.
2. If the range of supported guest driver versions does not overlap the range set by the administrator, the guest driver fails to load.
3. If the range of supported guest driver versions overlaps the range set by the administrator, the guest driver will load with a negotiated vGPU version that is the maximum value in the overlapping range.
4. No VMs must be running on the host when this function is called. If a VM is running on the host, the call to this function fails.

Parameters:

- vgpuVersion* Pointer to a caller-supplied range of supported vGPU versions.

Returns:

- [NVML_SUCCESS](#) The preset range of supported vGPU versions was successfully overridden.
- [NVML_ERROR_NOT_SUPPORTED](#) The API is not supported.
- [NVML_ERROR_IN_USE](#) The range was not overridden because a VM is running on the host.
- [NVML_ERROR_INVALID_ARGUMENT](#) The *vgpuVersion* parameter specifies a range that is outside the range supported by the NVIDIA vGPU Manager or if *vgpuVersion* is NULL.

6.29.3.6 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetMetadata (nvmlVgpuInstance_t vgpuInstance, nvmlVgpuMetadata_t * vgpuMetadata, unsigned int * bufferSize)`

Returns vGPU metadata structure for a running vGPU. The structure contains information about the vGPU and its associated VM such as the currently installed NVIDIA guest driver version, together with host driver version and an opaque data section containing internal state.

`nvmlVgpuInstanceGetMetadata()` may be called at any time for a vGPU instance. Some fields in the returned structure are dependent on information obtained from the guest VM, which may not yet have reached a state where that information is available. The current state of these dependent fields is reflected in the info structure's `nvmlVgpuGuestInfoState_t` field.

The VMM may choose to read and save the vGPU's VM info as persistent metadata associated with the VM, and provide it to 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*.

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

6.30 GRID Virtualization Utilization and Accounting

Functions

- `nvmlReturn_t DECLDIR nvmlDeviceGetVgpuUtilization` (`nvmlDevice_t device`, unsigned long long `lastSeenTimeStamp`, `nvmlValueType_t *sampleValType`, unsigned int `*vgpuInstanceSamplesCount`, `nvmlVgpuInstanceUtilizationSample_t *utilizationSamples`)
- `nvmlReturn_t DECLDIR nvmlDeviceGetVgpuProcessUtilization` (`nvmlDevice_t device`, unsigned long long `lastSeenTimeStamp`, unsigned int `*vgpuProcessSamplesCount`, `nvmlVgpuProcessUtilizationSample_t *utilizationSamples`)
- `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetAccountingMode` (`nvmlVgpuInstance_t vgpuInstance`, `nvmlEnableState_t *mode`)
- `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetAccountingPids` (`nvmlVgpuInstance_t vgpuInstance`, unsigned int `*count`, unsigned int `*pids`)
- `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetAccountingStats` (`nvmlVgpuInstance_t vgpuInstance`, unsigned int `pid`, `nvmlAccountingStats_t *stats`)
- `nvmlReturn_t DECLDIR nvmlVgpuInstanceClearAccountingPids` (`nvmlVgpuInstance_t vgpuInstance`)

6.30.1 Detailed Description

This chapter describes operations that are associated with vGPU Utilization and Accounting.

6.30.2 Function Documentation

6.30.2.1 `nvmlReturn_t DECLDIR nvmlDeviceGetVgpuProcessUtilization` (`nvmlDevice_t device`, unsigned long long `lastSeenTimeStamp`, unsigned int `*vgpuProcessSamplesCount`, `nvmlVgpuProcessUtilizationSample_t *utilizationSamples`)

Retrieves current utilization for processes running on vGPUs on a physical GPU (device).

For Maxwell TM or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder for processes running on vGPU instances active on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by *utilizationSamples*. One utilization sample structure is returned per process running on vGPU instances, that had some non-zero utilization during the last sample period. It includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "unsigned int" values.

To read utilization values, first determine the size of buffer required to hold the samples by invoking the function with *utilizationSamples* set to NULL. The function will return `NVML_ERROR_INSUFFICIENT_SIZE`, with the current vGPU instance count in *vgpuProcessSamplesCount*. The caller should allocate a buffer of size *vgpuProcessSamplesCount* * `sizeof(nvmlVgpuProcessUtilizationSample_t)`. Invoke the function again with the allocated buffer passed in *utilizationSamples*, and *vgpuProcessSamplesCount* set to the number of entries the buffer is sized for.

On successful return, the function updates *vgpuSubProcessSampleCount* with the number of vGPU sub process utilization sample structures that were actually written. This may differ from a previously read value depending on the number of processes that are active in any given sample period.

lastSeenTimeStamp represents the CPU timestamp in microseconds at which utilization samples were last read. Set it to 0 to read utilization based on all the samples maintained by the driver's internal sample buffer. Set *lastSeenTimeStamp* to a *timestamp* retrieved from a previous query to read utilization since the previous query.

Parameters:

device The identifier for the target device

lastSeenTimeStamp Return only samples with timestamp greater than lastSeenTimeStamp.

vgpuProcessSamplesCount Pointer to caller-supplied array size, and returns number of processes running on vGPU instances

utilizationSamples Pointer to caller-supplied buffer in which vGPU sub process utilization samples are returned

Returns:

- [NVML_SUCCESS](#) if utilization samples are successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* is invalid, *vgpuProcessSamplesCount* or a sample count of 0 is passed with a non-NULL *utilizationSamples*
- [NVML_ERROR_INSUFFICIENT_SIZE](#) if supplied *vgpuProcessSamplesCount* is too small to return samples for all vGPU instances currently executing on the device
- [NVML_ERROR_NOT_SUPPORTED](#) if vGPU is not supported by the device
- [NVML_ERROR_GPU_IS_LOST](#) if the target GPU has fallen off the bus or is otherwise inaccessible
- [NVML_ERROR_NOT_FOUND](#) if sample entries are not found
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.30.2.2 `nvmlReturn_t DECLDIR nvmlDeviceGetVgpuUtilization (nvmlDevice_t device, unsigned long long lastSeenTimeStamp, nvmlValueType_t * sampleValType, unsigned int * vgpuInstanceSamplesCount, nvmlVgpuInstanceUtilizationSample_t * utilizationSamples)`

Retrieves current utilization for vGPUs on a physical GPU (device).

For Kepler TM or newer fully supported devices.

Reads recent utilization of GPU SM (3D/Compute), framebuffer, video encoder, and video decoder for vGPU instances running on a device. Utilization values are returned as an array of utilization sample structures in the caller-supplied buffer pointed at by *utilizationSamples*. One utilization sample structure is returned per vGPU instance, and includes the CPU timestamp at which the samples were recorded. Individual utilization values are returned as "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.

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

6.30.2.3 `nvmlReturn_t DECLDIR nvmlVgpuInstanceClearAccountingPids (nvmlVgpuInstance_t vgpuInstance)`

Clears accounting information of the vGPU instance that have already terminated.

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

Note:

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

Only compute and graphics applications stats are reported and can be cleared since monitoring applications stats don't contribute to GPU utilization.

Parameters:

vgpuInstance The identifier of the target vGPU instance

Returns:

- [NVML_SUCCESS](#) if accounting information has been cleared
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is invalid
- [NVML_ERROR_NO_PERMISSION](#) if the user doesn't have permission to perform this operation
- [NVML_ERROR_NOT_SUPPORTED](#) if the vGPU doesn't support this feature or accounting mode is disabled
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.30.2.4 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetAccountingMode (nvmlVgpuInstance_t vgpuInstance, nvmlEnableState_t * mode)`

Queries the state of per process accounting mode on vGPU.

For Maxwell TM or newer fully supported devices.

Parameters:

vgpuInstance The identifier of the target vGPU instance
mode Reference in which to return the current accounting mode

Returns:

- [NVML_SUCCESS](#) if the mode has been successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *mode* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_NOT_SUPPORTED](#) if the vGPU doesn't support this feature
- [NVML_ERROR_DRIVER_NOT_LOADED](#) if NVIDIA driver is not running on the vGPU instance
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.30.2.5 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetAccountingPids (nvmlVgpuInstance_t vgpuInstance, unsigned int * count, unsigned int * pids)`

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

For Maxwell TM or newer fully supported devices.

To just query the maximum number of processes that can be queried, call this function with **count* = 0 and *pids*=NULL. The return code will be [NVML_ERROR_INSUFFICIENT_SIZE](#), or [NVML_SUCCESS](#) if list is empty.

For more details see [nvmlVgpuInstanceGetAccountingStats](#).

Note:

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

Parameters:

vgpuInstance The identifier of the target vGPU instance
count Reference in which to provide the *pids* array size, and to return the number of elements ready to be queried
pids Reference in which to return list of process ids

Returns:

- [NVML_SUCCESS](#) if *pids* were successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *count* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system
- [NVML_ERROR_NOT_SUPPORTED](#) if the vGPU doesn't support this feature or accounting mode is disabled

- [NVML_ERROR_INSUFFICIENT_SIZE](#) if *count* is too small (*count* is set to expected value)
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

See also:

[nvmlVgpuInstanceGetAccountingPids](#)

6.30.2.6 `nvmlReturn_t DECLDIR nvmlVgpuInstanceGetAccountingStats (nvmlVgpuInstance_t vgpuInstance, unsigned int pid, nvmlAccountingStats_t * stats)`

Queries process's accounting stats.

For Maxwell™ or newer fully supported devices.

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

See [nvmlAccountingStats_t](#) for description of each returned metric. List of processes that can be queried can be retrieved from [nvmlVgpuInstanceGetAccountingPids](#).

Note:

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

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

Parameters:

vgpuInstance The identifier of the target vGPU instance

pid Process Id of the target process to query stats for

stats Reference in which to return the process's accounting stats

Returns:

- [NVML_SUCCESS](#) if stats have been successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *vgpuInstance* is 0, or *stats* is NULL
- [NVML_ERROR_NOT_FOUND](#) if *vgpuInstance* does not match a valid active vGPU instance on the system or *stats* is not found
- [NVML_ERROR_NOT_SUPPORTED](#) if the vGPU doesn't support this feature or accounting mode is disabled
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.31 GPU Blacklist Queries

Data Structures

- struct [nvmlBlacklistDeviceInfo_t](#)

Functions

- [nvmlReturn_t](#) DECLDIR [nvmlGetBlacklistDeviceCount](#) (unsigned int *deviceCount)
- [nvmlReturn_t](#) DECLDIR [nvmlGetBlacklistDeviceInfoByIndex](#) (unsigned int index, [nvmlBlacklistDeviceInfo_t](#) *info)

6.31.1 Detailed Description

This chapter describes NVML operations that are associated with blacklisted GPUs.

6.31.2 Function Documentation

6.31.2.1 [nvmlReturn_t](#) DECLDIR [nvmlGetBlacklistDeviceCount](#) (unsigned int * *deviceCount*)

Retrieves the number of blacklisted GPU devices in the system.

For all products.

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

6.31.2.2 [nvmlReturn_t](#) DECLDIR [nvmlGetBlacklistDeviceInfoByIndex](#) (unsigned int *index*, [nvmlBlacklistDeviceInfo_t](#) * *info*)

Acquire the device information for a blacklisted device, based on its index.

For all products.

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.

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

See also:

[nvmlGetBlacklistDeviceCount](#)

6.32 Multi Instance GPU Management

Defines

- `#define NVML_DEVICE_MIG_DISABLE 0x0`
- `#define NVML_DEVICE_MIG_ENABLE 0x1`
- `#define NVML_GPU_INSTANCE_PROFILE_1_SLICE 0x0`
- `#define NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE 0x0`
- `#define NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_SHARED 0x0`

All the engines except multiprocessors would be shared.

Functions

- `nvmlReturn_t DECLDIR nvmlDeviceSetMigMode` (nvmlDevice_t device, unsigned int mode, nvmlReturn_t *activationStatus)
- `nvmlReturn_t DECLDIR nvmlDeviceGetMigMode` (nvmlDevice_t device, unsigned int *currentMode, unsigned int *pendingMode)
- `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstanceProfileInfo` (nvmlDevice_t device, unsigned int profile, nvmlGpuInstanceProfileInfo_t *info)
- `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstancePossiblePlacements` (nvmlDevice_t device, unsigned int profileId, nvmlGpuInstancePlacement_t *placements, unsigned int *count)
- `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstanceRemainingCapacity` (nvmlDevice_t device, unsigned int profileId, unsigned int *count)
- `nvmlReturn_t DECLDIR nvmlDeviceCreateGpuInstance` (nvmlDevice_t device, unsigned int profileId, nvmlGpuInstance_t *gpuInstance)
- `nvmlReturn_t DECLDIR nvmlGpuInstanceDestroy` (nvmlGpuInstance_t gpuInstance)
- `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstances` (nvmlDevice_t device, unsigned int profileId, nvmlGpuInstance_t *gpuInstances, unsigned int *count)
- `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstanceById` (nvmlDevice_t device, unsigned int id, nvmlGpuInstance_t *gpuInstance)
- `nvmlReturn_t DECLDIR nvmlGpuInstanceGetInfo` (nvmlGpuInstance_t gpuInstance, nvmlGpuInstanceInfo_t *info)
- `nvmlReturn_t DECLDIR nvmlGpuInstanceGetComputeInstanceProfileInfo` (nvmlGpuInstance_t gpuInstance, unsigned int profile, unsigned int engProfile, nvmlComputeInstanceProfileInfo_t *info)
- `nvmlReturn_t DECLDIR nvmlGpuInstanceGetComputeInstanceRemainingCapacity` (nvmlGpuInstance_t gpuInstance, unsigned int profileId, unsigned int *count)
- `nvmlReturn_t DECLDIR nvmlGpuInstanceCreateComputeInstance` (nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstance_t *computeInstance)
- `nvmlReturn_t DECLDIR nvmlComputeInstanceDestroy` (nvmlComputeInstance_t computeInstance)
- `nvmlReturn_t DECLDIR nvmlGpuInstanceGetComputeInstances` (nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstance_t *computeInstances, unsigned int *count)
- `nvmlReturn_t DECLDIR nvmlGpuInstanceGetComputeInstanceById` (nvmlGpuInstance_t gpuInstance, unsigned int id, nvmlComputeInstance_t *computeInstance)
- `nvmlReturn_t DECLDIR nvmlComputeInstanceGetInfo` (nvmlComputeInstance_t computeInstance, nvmlComputeInstanceInfo_t *info)
- `nvmlReturn_t DECLDIR nvmlDeviceIsMigDeviceHandle` (nvmlDevice_t device, unsigned int *isMigDevice)
- `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstanceId` (nvmlDevice_t device, unsigned int *id)
- `nvmlReturn_t DECLDIR nvmlDeviceGetComputeInstanceId` (nvmlDevice_t device, unsigned int *id)
- `nvmlReturn_t DECLDIR nvmlDeviceGetMaxMigDeviceCount` (nvmlDevice_t device, unsigned int *count)

- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetMigDeviceHandleByIndex](#) (nvmlDevice_t device, unsigned int index, nvmlDevice_t *migDevice)
- [nvmlReturn_t](#) DECLDIR [nvmlDeviceGetDeviceHandleFromMigDeviceHandle](#) (nvmlDevice_t migDevice, nvmlDevice_t *device)

6.32.1 Detailed Description

This chapter describes NVML operations that are associated with Multi Instance GPU management.

6.32.2 Define Documentation

6.32.2.1 #define NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE 0x0

Compute instance profiles.

These macros should be passed to [nvmlGpuInstanceGetComputeInstanceProfileInfo](#) to retrieve the detailed information about a compute instance such as profile ID, engine counts

6.32.2.2 #define NVML_DEVICE_MIG_DISABLE 0x0

Disable Multi Instance GPU mode.

6.32.2.3 #define NVML_DEVICE_MIG_ENABLE 0x1

Enable Multi Instance GPU mode.

6.32.2.4 #define NVML_GPU_INSTANCE_PROFILE_1_SLICE 0x0

GPU instance profiles.

These macros should be passed to [nvmlDeviceGetGpuInstanceProfileInfo](#) to retrieve the detailed information about a GPU instance such as profile ID, engine counts.

6.32.3 Function Documentation

6.32.3.1 nvmlReturn_t DECLDIR nvmlComputeInstanceDestroy (nvmlComputeInstance_t *computeInstance*)

Destroy compute instance.

For newer than Volta TM fully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

computeInstance The compute instance handle

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *computeInstance* is invalid
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

- [NVML_ERROR_IN_USE](#) If the compute instance is in use. This error would be returned if processes (e.g. CUDA application) are active on the compute instance.

6.32.3.2 `nvmlReturn_t DECLDIR nvmlComputeInstanceGetInfo (nvmlComputeInstance_t computeInstance, nvmlComputeInstanceInfo_t * info)`

Get compute instance information.

For newer than Volta TMfully supported devices. Supported on Linux only.

Parameters:

computeInstance The compute instance handle

info Return compute instance information

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *computeInstance* or *info* are invalid
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.32.3.3 `nvmlReturn_t DECLDIR nvmlDeviceCreateGpuInstance (nvmlDevice_t device, unsigned int profileId, nvmlGpuInstance_t * gpuInstance)`

Create GPU instance.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

If the parent device is unbound, reset or the GPU instance is destroyed explicitly, the GPU instance handle would become invalid. The GPU instance must be recreated to acquire a valid handle.

Parameters:

device The identifier of the target device

profileId The GPU instance profile ID. See [nvmlDeviceGetGpuInstanceProfileInfo](#)

gpuInstance Returns the GPU instance handle

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *profile*, *profileId* or *gpuInstance* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't have MIG mode enabled
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation
- [NVML_ERROR_INSUFFICIENT_RESOURCES](#) If the requested GPU instance could not be created

6.32.3.4 nvmlReturn_t DECLDIR nvmlDeviceGetComputeInstanceId (nvmlDevice_t *device*, unsigned int * *id*)

Get compute instance ID for the given MIG device handle.

Compute instance IDs are unique per GPU instance and remain valid until the compute instance is destroyed.

For newer than Volta TMfully supported devices. Supported on Linux only.

Parameters:

device Target MIG device handle

id Compute instance ID

Returns:

- [NVML_SUCCESS](#) if instance ID was successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* or *id* reference is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if this query is not supported by the device
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.32.3.5 nvmlReturn_t DECLDIR nvmlDeviceGetDeviceHandleFromMigDeviceHandle (nvmlDevice_t *migDevice*, nvmlDevice_t * *device*)

Get parent device handle from a MIG device handle.

For newer than Volta TMfully supported devices. Supported on Linux only.

Parameters:

migDevice MIG device handle

device Device handle

Returns:

- [NVML_SUCCESS](#) if *device* handle was successfully created
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *migDevice* or *device* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if this query is not supported by the device
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.32.3.6 nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstanceById (nvmlDevice_t *device*, unsigned int *id*, nvmlGpuInstance_t * *gpuInstance*)

Get GPU instances for given instance ID.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

device The identifier of the target device

id The GPU instance ID

gpuInstance Returns GPU instance

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *id* or *gpuInstance* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't have MIG mode enabled
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation
- [NVML_ERROR_NOT_FOUND](#) If the GPU instance is not found.

6.32.3.7 `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstanceId (nvmlDevice_t device, unsigned int * id)`

Get GPU instance ID for the given MIG device handle.

GPU instance IDs are unique per device and remain valid until the GPU instance is destroyed.

For newer than Volta TMfully supported devices. Supported on Linux only.

Parameters:

device Target MIG device handle

id GPU instance ID

Returns:

- [NVML_SUCCESS](#) if instance ID was successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* or *id* reference is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if this query is not supported by the device
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.32.3.8 `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstancePossiblePlacements (nvmlDevice_t device, unsigned int profileId, nvmlGpuInstancePlacement_t * placements, unsigned int * count)`

Get GPU instance placements.

A placement represents the location of a GPU instance within a device. This API only returns all the possible placements for the given profile.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

device The identifier of the target device

profileId The GPU instance profile ID. See [nvmlDeviceGetGpuInstanceProfileInfo](#)

placements Returns placements, the buffer must be large enough to accommodate the instances supported by the profile. See [nvmlDeviceGetGpuInstanceProfileInfo](#)

count The count of returned placements

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *profileId*, *placements* or *count* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't have MIG mode enabled or *profileId* isn't supported
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.32.3.9 `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstanceProfileInfo (nvmlDevice_t device, unsigned int profile, nvmlGpuInstanceProfileInfo_t * info)`

Get GPU instance profile information.

Information provided by this API is immutable throughout the lifetime of a MIG mode.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

device The identifier of the target device
profile One of the NVML_GPU_INSTANCE_PROFILE_*
info Returns detailed profile information

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *profile* or *info* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't have MIG mode enabled or *profile* isn't supported
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.32.3.10 `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstanceRemainingCapacity (nvmlDevice_t device, unsigned int profileId, unsigned int * count)`

Get GPU instance profile capacity.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

device The identifier of the target device
profileId The GPU instance profile ID. See [nvmlDeviceGetGpuInstanceProfileInfo](#)
count Returns remaining instance count for the profile ID

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *profileId* or *count* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't have MIG mode enabled or *profileId* isn't supported
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.32.3.11 `nvmlReturn_t DECLDIR nvmlDeviceGetGpuInstances (nvmlDevice_t device, unsigned int profileId, nvmlGpuInstance_t * gpuInstances, unsigned int * count)`

Get GPU instances for given profile ID.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

device The identifier of the target device

profileId The GPU instance profile ID. See [nvmlDeviceGetGpuInstanceProfileInfo](#)

gpuInstances Returns pre-existing GPU instances, the buffer must be large enough to accommodate the instances supported by the profile. See [nvmlDeviceGetGpuInstanceProfileInfo](#)

count The count of returned GPU instances

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *profileId*, *gpuInstances* or *count* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't have MIG mode enabled
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.32.3.12 `nvmlReturn_t DECLDIR nvmlDeviceGetMaxMigDeviceCount (nvmlDevice_t device, unsigned int * count)`

Get the maximum number of MIG devices that can exist under a given parent NVML device.

Returns zero if MIG is not supported or enabled.

For newer than Volta TMfully supported devices. Supported on Linux only.

Parameters:

device Target device handle

count Count of MIG devices

Returns:

- [NVML_SUCCESS](#) if *count* was successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* or *count* reference is invalid
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.32.3.13 `nvmlReturn_t DECLDIR nvmlDeviceGetMigDeviceHandleByIndex (nvmlDevice_t device, unsigned int index, nvmlDevice_t * migDevice)`

Get MIG device handle for the given index under its parent NVML device.

If the compute instance is destroyed either explicitly or by destroying, resetting or unbinding the parent GPU instance or the GPU device itself the MIG device handle would remain invalid and must be requested again using this API. Handles may be reused and their properties can change in the process.

For newer than Volta TMfully supported devices. Supported on Linux only.

Parameters:

device Reference to the parent GPU device handle

index Index of the MIG device

migDevice Reference to the MIG device handle

Returns:

- [NVML_SUCCESS](#) if *migDevice* handle was successfully created
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device*, *index* or *migDevice* reference is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if this query is not supported by the device
- [NVML_ERROR_NOT_FOUND](#) if no valid MIG device was found at *index*
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.32.3.14 `nvmlReturn_t DECLDIR nvmlDeviceGetMigMode (nvmlDevice_t device, unsigned int * currentMode, unsigned int * pendingMode)`

Get MIG mode for the device.

For newer than Volta TM fully supported devices. Supported on Linux only.

Changing MIG modes may require device unbind or reset. The "pending" MIG mode refers to the target mode following the next activation trigger.

Parameters:

device The identifier of the target device

currentMode Returns the current mode, [NVML_DEVICE_MIG_DISABLE](#) or [NVML_DEVICE_MIG_ENABLE](#)

pendingMode Returns the pending mode, [NVML_DEVICE_MIG_DISABLE](#) or [NVML_DEVICE_MIG_ENABLE](#)

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *currentMode* or *pendingMode* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't support MIG mode

6.32.3.15 `nvmlReturn_t DECLDIR nvmlDeviceIsMigDeviceHandle (nvmlDevice_t device, unsigned int * isMigDevice)`

Test if the given handle refers to a MIG device.

A MIG device handle is an NVML abstraction which maps to a MIG compute instance. These overloaded references can be used (with some restrictions) interchangeably with a GPU device handle to execute queries at a per-compute instance granularity.

For newer than Volta TM fully supported devices. Supported on Linux only.

Parameters:

device NVML handle to test

isMigDevice True when handle refers to a MIG device

Returns:

- [NVML_SUCCESS](#) if *device* status was successfully retrieved
- [NVML_ERROR_UNINITIALIZED](#) if the library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) if *device* handle or *isMigDevice* reference is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) if this check is not supported by the device
- [NVML_ERROR_UNKNOWN](#) on any unexpected error

6.32.3.16 `nvmlReturn_t DECLDIR nvmlDeviceSetMigMode (nvmlDevice_t device, unsigned int mode, nvmlReturn_t * activationStatus)`

Set MIG mode for the device.

For newer than Volta TM fully supported devices. Supported on Linux only. Requires root user.

This mode determines whether a GPU instance can be created.

This API may unbind or reset the device to activate the requested mode. Thus, the attributes associated with the device, such as minor number, might change. The caller of this API is expected to query such attributes again.

On certain platforms like pass-through virtualization, where reset functionality may not be exposed directly, VM reboot is required. *activationStatus* would return [NVML_ERROR_RESET_REQUIRED](#) for such cases.

activationStatus would return the appropriate error code upon unsuccessful activation. For example, if device unbind fails because the device isn't idle, [NVML_ERROR_IN_USE](#) would be returned. The caller of this API is expected to idle the device and retry setting the *mode*.

Parameters:

device The identifier of the target device

mode The mode to be set, [NVML_DEVICE_MIG_DISABLE](#) or [NVML_DEVICE_MIG_ENABLE](#)

activationStatus The activationStatus status

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *mode* or *activationStatus* are invalid
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't support MIG mode

6.32.3.17 `nvmlReturn_t DECLDIR nvmlGpuInstanceCreateComputeInstance (nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstance_t * computeInstance)`

Create compute instance.

For newer than Volta TM fully supported devices. Supported on Linux only. Requires privileged user.

If the parent device is unbound, reset or the parent GPU instance is destroyed or the compute instance is destroyed explicitly, the compute instance handle would become invalid. The compute instance must be recreated to acquire a valid handle.

Parameters:

gpuInstance The identifier of the target GPU instance
profileId The compute instance profile ID. See [nvmlGpuInstanceGetComputeInstanceProfileInfo](#)
computeInstance Returns the compute instance handle

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *gpuInstance*, *profile*, *profileId* or *computeInstance* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *profileId* isn't supported
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation
- [NVML_ERROR_INSUFFICIENT_RESOURCES](#) If the requested compute instance could not be created

6.32.3.18 nvmlReturn_t DECLDIR nvmlGpuInstanceDestroy (nvmlGpuInstance_t *gpuInstance*)

Destroy GPU instance.

For newer than Volta™ fully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

gpuInstance The GPU instance handle

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *gpuInstance* is invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't have MIG mode enabled
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation
- [NVML_ERROR_IN_USE](#) If the GPU instance is in use. This error would be returned if processes (e.g. CUDA application) or compute instances are active on the GPU instance.

6.32.3.19 nvmlReturn_t DECLDIR nvmlGpuInstanceGetComputeInstanceById (nvmlGpuInstance_t *gpuInstance*, unsigned int *id*, nvmlComputeInstance_t * *computeInstance*)

Get compute instance for given instance ID.

For newer than Volta™ fully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

gpuInstance The identifier of the target GPU instance
id The compute instance ID
computeInstance Returns compute instance

Returns:

- [NVML_SUCCESS](#) Upon success

- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *device*, *ID* or *computeInstance* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *device* doesn't have MIG mode enabled
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation
- [NVML_ERROR_NOT_FOUND](#) If the compute instance is not found.

6.32.3.20 `nvmlReturn_t DECLDIR nvmlGpuInstanceGetComputeInstanceProfileInfo (nvmlGpuInstance_t gpuInstance, unsigned int profile, unsigned int engProfile, nvmlComputeInstanceProfileInfo_t * info)`

Get compute instance profile information.

Information provided by this API is immutable throughout the lifetime of a MIG mode.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

gpuInstance The identifier of the target GPU instance
profile One of the NVML_COMPUTE_INSTANCE_PROFILE_*
engProfile One of the NVML_COMPUTE_INSTANCE_ENGINE_PROFILE_*
info Returns detailed profile information

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *gpuInstance*, *profile*, *engProfile* or *info* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *profile* isn't supported
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.32.3.21 `nvmlReturn_t DECLDIR nvmlGpuInstanceGetComputeInstanceRemainingCapacity (nvmlGpuInstance_t gpuInstance, unsigned int profileId, unsigned int * count)`

Get compute instance profile capacity.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

gpuInstance The identifier of the target GPU instance
profileId The compute instance profile ID. See [nvmlGpuInstanceGetComputeInstanceProfileInfo](#)
count Returns remaining instance count for the profile ID

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *gpuInstance*, *profileId* or *availableCount* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *profileId* isn't supported
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.32.3.22 `nvmlReturn_t DECLDIR nvmlGpuInstanceGetComputeInstances (nvmlGpuInstance_t gpuInstance, unsigned int profileId, nvmlComputeInstance_t * computeInstances, unsigned int * count)`

Get compute instances for given profile ID.

For newer than Volta TMfully supported devices. Supported on Linux only. Requires privileged user.

Parameters:

gpuInstance The identifier of the target GPU instance

profileId The compute instance profile ID. See [nvmlGpuInstanceGetComputeInstanceProfileInfo](#)

computeInstances Returns pre-existing compute instances, the buffer must be large enough to accommodate the instances supported by the profile. See [nvmlGpuInstanceGetComputeInstanceProfileInfo](#)

count The count of returned compute instances

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *gpuInstance*, *profileId*, *computeInstances* or *count* are invalid
- [NVML_ERROR_NOT_SUPPORTED](#) If *profileId* isn't supported
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.32.3.23 `nvmlReturn_t DECLDIR nvmlGpuInstanceGetInfo (nvmlGpuInstance_t gpuInstance, nvmlGpuInstanceInfo_t * info)`

Get GPU instance information.

For newer than Volta TMfully supported devices. Supported on Linux only.

Parameters:

gpuInstance The GPU instance handle

info Return GPU instance information

Returns:

- [NVML_SUCCESS](#) Upon success
- [NVML_ERROR_UNINITIALIZED](#) If library has not been successfully initialized
- [NVML_ERROR_INVALID_ARGUMENT](#) If *gpuInstance* or *info* are invalid
- [NVML_ERROR_NO_PERMISSION](#) If user doesn't have permission to perform the operation

6.33 NvmlClocksThrottleReasons

Defines

- `#define nvmlClocksThrottleReasonGpuIdle 0x0000000000000001LL`
- `#define nvmlClocksThrottleReasonApplicationsClocksSetting 0x0000000000000002LL`
- `#define nvmlClocksThrottleReasonUserDefinedClocks nvmlClocksThrottleReasonApplicationsClocksSetting`
- `#define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL`
- `#define nvmlClocksThrottleReasonHwSlowdown 0x0000000000000008LL`
- `#define nvmlClocksThrottleReasonSyncBoost 0x0000000000000010LL`
- `#define nvmlClocksThrottleReasonSwThermalSlowdown 0x0000000000000020LL`
- `#define nvmlClocksThrottleReasonHwThermalSlowdown 0x0000000000000040LL`
- `#define nvmlClocksThrottleReasonHwPowerBrakeSlowdown 0x0000000000000080LL`
- `#define nvmlClocksThrottleReasonDisplayClockSetting 0x0000000000000100LL`
- `#define nvmlClocksThrottleReasonNone 0x0000000000000000LL`
- `#define nvmlClocksThrottleReasonAll`

6.33.1 Define Documentation

6.33.1.1 `#define nvmlClocksThrottleReasonAll`

Value:

```
(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

6.33.1.2 `#define nvmlClocksThrottleReasonApplicationsClocksSetting 0x0000000000000002LL`

GPU clocks are limited by current setting of applications clocks

See also:

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

6.33.1.3 `#define nvmlClocksThrottleReasonDisplayClockSetting 0x0000000000000100LL`

GPU clocks are limited by current setting of Display clocks

See also:

bug 1997531

6.33.1.4 #define nvmlClocksThrottleReasonGpuIdle 0x0000000000000001LL

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

Note:

This limiter may be removed in a later release

6.33.1.5 #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](#)

6.33.1.6 #define nvmlClocksThrottleReasonHwSlowdown 0x0000000000000008LL

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

This is an indicator of:

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

See also:

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

6.33.1.7 #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](#)

6.33.1.8 #define nvmlClocksThrottleReasonNone 0x0000000000000000LL

Bit mask representing no clocks throttling

Clocks are as high as possible.

6.33.1.9 #define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL

SW Power Scaling algorithm is reducing the clocks below requested clocks

See also:

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

6.33.1.10 #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

6.33.1.11 #define nvmlClocksThrottleReasonSyncBoost 0x0000000000000010LL

Sync Boost

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

6.33.1.12 #define nvmlClocksThrottleReasonUserDefinedClocks nvmlClocksThrottleReasonApplication-sClocksSetting

Deprecated

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

Chapter 7

Data Structure Documentation

7.1 nvmlAccountingStats_t Struct Reference

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

Data Fields

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

7.1.1 Detailed Description

Describes accounting statistics of a process.

7.2 nvmlBAR1Memory_t Struct Reference

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

Data Fields

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

7.2.1 Detailed Description

BAR1 Memory allocation Information for a device

7.3 nvmlBlacklistDeviceInfo_t Struct Reference

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

Data Fields

- [nvmlPciInfo_t pciInfo](#)
The PCI information for the blacklisted GPU.
- char [uuid](#) [NVML_DEVICE_UUID_BUFFER_SIZE]
The ASCII string UUID for the blacklisted GPU.

7.3.1 Detailed Description

Blacklist GPU device information

7.4 nvmlBridgeChipHierarchy_t Struct Reference

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

Data Fields

- unsigned char [bridgeCount](#)
Number of Bridge Chips on the Board.
- [nvmlBridgeChipInfo_t bridgeChipInfo](#) [NVML_MAX_PHYSICAL_BRIDGE]
Hierarchy of Bridge Chips on the board.

7.4.1 Detailed Description

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

7.5 nvmlBridgeChipInfo_t Struct Reference

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

Data Fields

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

7.5.1 Detailed Description

Information about the Bridge Chip Firmware

7.6 nvmlEccErrorCounts_t Struct Reference

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

Data Fields

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

7.6.1 Detailed Description

Detailed ECC error counts for a device.

Deprecated

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

7.7 nvmlEncoderSessionInfo_t Struct Reference

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

Data Fields

- unsigned int [sessionId](#)
Unique session ID.
- unsigned int [pid](#)
Owning process ID.
- nvmlVgpuInstance_t [vgpuInstance](#)
Owning vGPU instance ID (only valid on vGPU hosts, otherwise zero).
- nvmlEncoderType_t [codecType](#)
Video encoder type.
- unsigned int [hResolution](#)
Current encode horizontal resolution.
- unsigned int [vResolution](#)
Current encode vertical resolution.
- unsigned int [averageFps](#)
Moving average encode frames per second.
- unsigned int [averageLatency](#)
Moving average encode latency in microseconds.

7.7.1 Detailed Description

Structure to hold encoder session data

7.8 nvmlEventData_t Struct Reference

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

Data Fields

- nvmlDevice_t [device](#)
Specific device where the event occurred.
- unsigned long long [eventType](#)
Information about what specific event occurred.
- unsigned long long [eventData](#)
Stores XID error for the device in the event of nvmlEventTypeXidCriticalError,.
- unsigned int [gpuInstanceId](#)
If MIG is enabled and nvmlEventTypeXidCriticalError event is attributable to a GPU.
- unsigned int [computeInstanceId](#)
If MIG is enabled and nvmlEventTypeXidCriticalError event is attributable to a.

7.8.1 Detailed Description

Information about occurred event

7.9 nvmlFBCSessionInfo_t Struct Reference

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

Data Fields

- unsigned int [sessionId](#)
Unique session ID.
- unsigned int [pid](#)
Owning process ID.
- nvmlVgpuInstance_t [vgpuInstance](#)
Owning vGPU instance ID (only valid on vGPU hosts, otherwise zero).
- unsigned int [displayOrdinal](#)
Display identifier.
- [nvmlFBCSessionType_t](#) [sessionType](#)
Type of frame buffer capture session.
- unsigned int [sessionFlags](#)
Session flags (one or more of NVML_NVFBC_SESSION_FLAG_XXX).
- unsigned int [hMaxResolution](#)
Max horizontal resolution supported by the capture session.
- unsigned int [vMaxResolution](#)
Max vertical resolution supported by the capture session.
- unsigned int [hResolution](#)
Horizontal resolution requested by caller in capture call.
- unsigned int [vResolution](#)
Vertical resolution requested by caller in capture call.
- unsigned int [averageFPS](#)
Moving average new frames captured per second.
- unsigned int [averageLatency](#)
Moving average new frame capture latency in microseconds.

7.9.1 Detailed Description

Structure to hold FBC session data

7.10 nvmlFBCStats_t Struct Reference

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

Data Fields

- unsigned int [sessionsCount](#)
Total no of sessions.
- unsigned int [averageFPS](#)
Moving average new frames captured per second.
- unsigned int [averageLatency](#)
Moving average new frame capture latency in microseconds.

7.10.1 Detailed Description

Structure to hold frame buffer capture sessions stats

7.11 nvmlFieldValue_t Struct Reference

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

Data Fields

- unsigned int [fieldId](#)
ID of the NVML field to retrieve. This must be set before any call that uses this struct. See the constants starting with NVML_FI_ above.
- unsigned int [scopeId](#)
Scope ID can represent data used by NVML depending on fieldId's context. For example, for NVLink throughput counter data, scopeId can represent linkId.
- long long [timestamp](#)
CPU Timestamp of this value in microseconds since 1970.
- long long [latencyUsec](#)
How long this field value took to update (in usec) within NVML. This may be averaged across several fields that are serviced by the same driver call.
- [nvmlValueType_t](#) [valueType](#)
Type of the value stored in value.
- [nvmlReturn_t](#) [nvmlReturn](#)
Return code for retrieving this value. This must be checked before looking at value, as value is undefined if nvmlReturn != NVML_SUCCESS.
- [nvmlValue_t](#) [value](#)
Value for this field. This is only valid if nvmlReturn == NVML_SUCCESS.

7.11.1 Detailed Description

Information for a Field Value Sample

7.12 nvmlGridLicensableFeature_t Struct Reference

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

Data Fields

- [nvmlGridLicenseFeatureCode_t featureCode](#)
Licensed feature code.
- unsigned int [featureState](#)
Non-zero if feature is currently licensed, otherwise zero.
- unsigned int [featureEnabled](#)
Non-zero if feature is enabled, otherwise zero.

7.12.1 Detailed Description

Structure containing GRID licensable feature information

7.13 nvmlGridLicensableFeatures_t Struct Reference

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

Data Fields

- int [isGridLicenseSupported](#)
Non-zero if GRID Software Licensing is supported on the system, otherwise zero.
- unsigned int [licensableFeaturesCount](#)
Entries returned in gridLicensableFeatures array.
- [nvmlGridLicensableFeature_t](#) [gridLicensableFeatures](#) [NVML_GRID_LICENSE_FEATURE_MAX_COUNT]
Array of GRID licensable features.

7.13.1 Detailed Description

Structure to store GRID licensable features

7.14 nvmlHwbcEntry_t Struct Reference

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

7.14.1 Detailed Description

Description of HWBC entry

7.15 nvmlLedState_t Struct Reference

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

Data Fields

- char `cause` [256]
If amber, a text description of the cause.
- `nvmlLedColor_t` `color`
GREEN or AMBER.

7.15.1 Detailed Description

LED states for an S-class unit.

7.16 nvmlMemory_t Struct Reference

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

Data Fields

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

7.16.1 Detailed Description

Memory allocation information for a device.

7.17 nvmlNvLinkUtilizationControl_t Struct Reference

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

7.17.1 Detailed Description

Struct to define the NVLINK counter controls

7.18 nvmlPciInfo_t Struct Reference

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

Data Fields

- char [busIdLegacy](#) [NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE]
The legacy tuple domain:bus:device:function PCI identifier (& NULL terminator).
- unsigned int [domain](#)
The PCI domain on which the device's bus resides, 0 to 0xffffffff.
- unsigned int [bus](#)
The bus on which the device resides, 0 to 0xff.
- unsigned int [device](#)
The device's id on the bus, 0 to 31.
- unsigned int [pciDeviceId](#)
The combined 16-bit device id and 16-bit vendor id.
- unsigned int [pciSubSystemId](#)
The 32-bit Sub System Device ID.
- char [busId](#) [NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE]
The tuple domain:bus:device:function PCI identifier (& NULL terminator).

7.18.1 Detailed Description

PCI information about a GPU device.

7.19 nvmlProcessInfo_t Struct Reference

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

Data Fields

- unsigned int [pid](#)
Process ID.
- unsigned long long [usedGpuMemory](#)
Amount of used GPU memory in bytes. Under WDDM, [NVML_VALUE_NOT_AVAILABLE](#) is always reported because Windows KMD manages all the memory and not the NVIDIA driver.
- unsigned int [gpuInstanceId](#)
If MIG is enabled, stores a valid GPU instance ID. [gpuInstanceId](#) is set to.
- unsigned int [computeInstanceId](#)
If MIG is enabled, stores a valid compute instance ID. [computeInstanceId](#) is set to.

7.19.1 Detailed Description

Information about running compute processes on the GPU

7.20 nvmlProcessUtilizationSample_t Struct Reference

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

Data Fields

- unsigned int [pid](#)
PID of process.
- unsigned long long [timeStamp](#)
CPU Timestamp in microseconds.
- unsigned int [smUtil](#)
SM (3D/Compute) Util Value.
- unsigned int [memUtil](#)
Frame Buffer Memory Util Value.
- unsigned int [encUtil](#)
Encoder Util Value.
- unsigned int [decUtil](#)
Decoder Util Value.

7.20.1 Detailed Description

Structure to store utilization value and process Id

7.21 nvmlPSUInfo_t Struct Reference

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

Data Fields

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

7.21.1 Detailed Description

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

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

7.22 nvmlRowRemapperHistogramValues_t Struct Reference

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

7.22.1 Detailed Description

Possible values that classify the remap availability for each bank. The max field will contain the number of banks that have maximum remap availability (all reserved rows are available). None means that there are no reserved rows available.

7.23 nvmlSample_t Struct Reference

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

Data Fields

- unsigned long long [timeStamp](#)
CPU Timestamp in microseconds.
- [nvmlValue_t](#) [sampleValue](#)
Sample Value.

7.23.1 Detailed Description

Information for Sample

7.24 nvmlUnitFanInfo_t Struct Reference

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

Data Fields

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

7.24.1 Detailed Description

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

7.25 nvmlUnitFanSpeeds_t Struct Reference

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

Data Fields

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

7.25.1 Detailed Description

Fan speed readings for an entire S-class unit.

7.26 nvmlUnitInfo_t Struct Reference

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

Data Fields

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

7.26.1 Detailed Description

Static S-class unit info.

7.27 nvmlUtilization_t Struct Reference

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

Data Fields

- unsigned int [gpu](#)
Percent of time over the past sample period during which one or more kernels was executing on the GPU.
- unsigned int [memory](#)
Percent of time over the past sample period during which global (device) memory was being read or written.

7.27.1 Detailed Description

Utilization information for a device. Each sample period may be between 1 second and 1/6 second, depending on the product being queried.

7.28 nvmlValue_t Union Reference

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

Data Fields

- double [dVal](#)
If the value is double.
- unsigned int [uiVal](#)
If the value is unsigned int.
- unsigned long [ulVal](#)
If the value is unsigned long.
- unsigned long long [ullVal](#)
If the value is unsigned long long.
- signed long long [sllVal](#)
If the value is signed long long.

7.28.1 Detailed Description

Union to represent different types of Value

7.29 nvmlVgpuInstanceUtilizationSample_t Struct Reference

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

Data Fields

- `nvmlVgpuInstance_t vgpuInstance`
vGPU Instance
- `unsigned long long timeStamp`
CPU Timestamp in microseconds.
- `nvmlValue_t smUtil`
SM (3D/Compute) Util Value.
- `nvmlValue_t memUtil`
Frame Buffer Memory Util Value.
- `nvmlValue_t encUtil`
Encoder Util Value.
- `nvmlValue_t decUtil`
Decoder Util Value.

7.29.1 Detailed Description

Structure to store Utilization Value and vgpuInstance

7.30 nvmlVgpuMetadata_t Struct Reference

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

Data Fields

- unsigned int [version](#)
Current version of the structure.
- unsigned int [revision](#)
Current revision of the structure.
- [nvmlVgpuGuestInfoState_t](#) [guestInfoState](#)
Current state of Guest-dependent fields.
- char [guestDriverVersion](#) [NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE]
Version of driver installed in guest.
- char [hostDriverVersion](#) [NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE]
Version of driver installed in host.
- unsigned int [reserved](#) [6]
Reserved for internal use.
- unsigned int [vgpuVirtualizationCaps](#)
vGPU virtualizaion capabilities bitfileld
- unsigned int [guestVgpuVersion](#)
vGPU version of guest driver
- unsigned int [opaqueDataSize](#)
Size of opaque data field in bytes.
- char [opaqueData](#) [4]
Opaque data.

7.30.1 Detailed Description

vGPU metadata structure.

7.31 `nvmlVgpuPgpuCompatibility_t` Struct Reference

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

Data Fields

- [nvmlVgpuVmCompatibility_t vgpuVmCompatibility](#)
Compatibility of vGPU VM. See [nvmlVgpuVmCompatibility_t](#).
- [nvmlVgpuPgpuCompatibilityLimitCode_t compatibilityLimitCode](#)
Limiting factor for vGPU-pGPU compatibility. See [nvmlVgpuPgpuCompatibilityLimitCode_t](#).

7.31.1 Detailed Description

vGPU-pGPU compatibility structure

7.32 nvmlVgpuPgpuMetadata_t Struct Reference

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

Data Fields

- unsigned int [version](#)
Current version of the structure.
- unsigned int [revision](#)
Current revision of the structure.
- char [hostDriverVersion](#) [NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE]
Host driver version.
- unsigned int [pgpuVirtualizationCaps](#)
Pgpu virtualizaion capabilities bitfield.
- unsigned int [reserved](#) [5]
Reserved for internal use.
- [nvmlVgpuVersion_t](#) [hostSupportedVgpuRange](#)
vGPU version range supported by host driver
- unsigned int [opaqueDataSize](#)
Size of opaque data field in bytes.
- char [opaqueData](#) [4]
Opaque data.

7.32.1 Detailed Description

Physical GPU metadata structure

7.33 nvmlVgpuProcessUtilizationSample_t Struct Reference

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

Data Fields

- nvmlVgpuInstance_t [vgpuInstance](#)
vGPU Instance
- unsigned int [pid](#)
PID of process running within the vGPU VM.
- char [processName](#) [NVML_VGPU_NAME_BUFFER_SIZE]
Name of process running within the vGPU VM.
- unsigned long long [timeStamp](#)
CPU Timestamp in microseconds.
- unsigned int [smUtil](#)
SM (3D/Compute) Util Value.
- unsigned int [memUtil](#)
Frame Buffer Memory Util Value.
- unsigned int [encUtil](#)
Encoder Util Value.
- unsigned int [decUtil](#)
Decoder Util Value.

7.33.1 Detailed Description

Structure to store Utilization Value, vgpuInstance and subprocess information

7.34 nvmlVgpuVersion_t Struct Reference

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

Data Fields

- unsigned int [minVersion](#)
Minimum vGPU version.
- unsigned int [maxVersion](#)
Maximum vGPU version.

7.34.1 Detailed Description

Structure representing range of vGPU versions.

7.35 nvmlViolationTime_t Struct Reference

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

Data Fields

- unsigned long long [referenceTime](#)
referenceTime represents CPU timestamp in microseconds
- unsigned long long [violationTime](#)
violationTime in Nanoseconds

7.35.1 Detailed Description

Struct to hold perf policy violation status data

Index

Accounting Statistics, [45](#)

Constants, [55](#)

CPU and Memory Affinity, [118](#)

definitions related to the drain state, [51](#)

Device Commands, [123](#)

Device Enums, [19](#)

Device Queries, [64](#)

Device Structs, [15](#)

Drain states, [142](#)

Encoder Structs, [49](#)

Error reporting, [54](#)

Event Handling Methods, [138](#)

Event Types, [43](#)

Field Value Enums, [32](#)

Field Value Queries, [145](#)

Frame Buffer Capture Structures, [50](#)

GPU Blacklist Queries, [174](#)

GRID vGPU Management, [150](#)

GRID Virtualization APIs, [147](#)

GRID Virtualization Constants, [30](#)

GRID Virtualization Enums, [28](#)

GRID Virtualization Enums, Constants and Structs, [146](#)

GRID Virtualization Migration, [164](#)

GRID Virtualization Structs, [31](#)

GRID Virtualization Utilization and Accounting, [169](#)

Initialization and Cleanup, [52](#)

Multi Instance GPU Management, [176](#)

NvLink

 nvmlDeviceFreezeNvLinkUtilizationCounter, [132](#)

 nvmlDeviceGetNvLinkCapability, [133](#)

 nvmlDeviceGetNvLinkErrorCounter, [133](#)

 nvmlDeviceGetNvLinkRemotePciInfo_v2, [133](#)

 nvmlDeviceGetNvLinkState, [134](#)

 nvmlDeviceGetNvLinkUtilizationControl, [134](#)

 nvmlDeviceGetNvLinkUtilizationCounter, [135](#)

 nvmlDeviceGetNvLinkVersion, [135](#)

 nvmlDeviceResetNvLinkErrorCounters, [136](#)

 nvmlDeviceResetNvLinkUtilizationCounter, [136](#)

 nvmlDeviceSetNvLinkUtilizationControl, [137](#)

NvLink Methods, [132](#)

nvml

 NVML_VGPU_COMPATIBILITY_LIMIT_GPU, [165](#)

 NVML_VGPU_COMPATIBILITY_LIMIT_GUEST_DRIVER, [165](#)

 NVML_VGPU_COMPATIBILITY_LIMIT_HOST_DRIVER, [165](#)

 NVML_VGPU_COMPATIBILITY_LIMIT_NONE, [165](#)

 NVML_VGPU_COMPATIBILITY_LIMIT_OTHER, [165](#)

 NVML_VGPU_VM_COMPATIBILITY_COLD, [165](#)

 NVML_VGPU_VM_COMPATIBILITY_HIBERNATE, [165](#)

 NVML_VGPU_VM_COMPATIBILITY_LIVE, [165](#)

 NVML_VGPU_VM_COMPATIBILITY_NONE, [165](#)

 NVML_VGPU_VM_COMPATIBILITY_SLEEP, [165](#)

 nvmlDeviceGetPgpuMetadataString, [165](#)

 nvmlDeviceGetVgpuMetadata, [165](#)

 nvmlGetVgpuCompatibility, [166](#)

 nvmlGetVgpuVersion, [166](#)

 nvmlSetVgpuVersion, [167](#)

 nvmlVgpuInstanceGetMetadata, [168](#)

 nvmlVgpuPgpuCompatibilityLimitCode_t, [165](#)

 nvmlVgpuVmCompatibility_t, [165](#)

 NVML_AGGREGATE_ECC

 nvmlDeviceEnumvs, [23](#)

 NVML_CLOCK_COUNT

 nvmlDeviceEnumvs, [22](#)

 NVML_CLOCK_GRAPHICS

 nvmlDeviceEnumvs, [22](#)

 NVML_CLOCK_ID_APP_CLOCK_DEFAULT

 nvmlDeviceEnumvs, [22](#)

 NVML_CLOCK_ID_APP_CLOCK_TARGET

 nvmlDeviceEnumvs, [22](#)

 NVML_CLOCK_ID_COUNT

 nvmlDeviceEnumvs, [22](#)

 NVML_CLOCK_ID_CURRENT

 nvmlDeviceEnumvs, [22](#)

 NVML_CLOCK_ID_CUSTOMER_BOOST_MAX

- nvmlDeviceEnumvs, [22](#)
- NVML_CLOCK_MEM
 - nvmlDeviceEnumvs, [22](#)
- NVML_CLOCK_SM
 - nvmlDeviceEnumvs, [22](#)
- NVML_CLOCK_VIDEO
 - nvmlDeviceEnumvs, [22](#)
- NVML_COMPUTEMODE_DEFAULT
 - nvmlDeviceEnumvs, [23](#)
- NVML_COMPUTEMODE_EXCLUSIVE_PROCESS
 - nvmlDeviceEnumvs, [23](#)
- NVML_COMPUTEMODE_EXCLUSIVE_THREAD
 - nvmlDeviceEnumvs, [23](#)
- NVML_COMPUTEMODE_PROHIBITED
 - nvmlDeviceEnumvs, [23](#)
- NVML_DEC_UTILIZATION_SAMPLES
 - nvmlDeviceStructs, [18](#)
- NVML_DRIVER_WDDM
 - nvmlDeviceEnumvs, [23](#)
- NVML_DRIVER_WDM
 - nvmlDeviceEnumvs, [23](#)
- NVML_ECC_COUNTER_TYPE_COUNT
 - nvmlDeviceEnumvs, [23](#)
- NVML_ENC_UTILIZATION_SAMPLES
 - nvmlDeviceStructs, [18](#)
- NVML_ENCODER_QUERY_H264
 - nvmlEncoderStructs, [49](#)
- NVML_ENCODER_QUERY_HEVC
 - nvmlEncoderStructs, [49](#)
- NVML_ERROR_ALREADY_INITIALIZED
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_CORRUPTED_INFOROM
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_DRIVER_NOT_LOADED
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_FUNCTION_NOT_FOUND
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_GPU_IS_LOST
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_IN_USE
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_INSUFFICIENT_POWER
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_INSUFFICIENT_RESOURCES
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_INSUFFICIENT_SIZE
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_INVALID_ARGUMENT
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_IRQ_ISSUE
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_LIB_RM_VERSION_MISMATCH
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_LIBRARY_NOT_FOUND
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_MEMORY
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_NO_DATA
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_NO_PERMISSION
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_NOT_FOUND
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_NOT_SUPPORTED
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_OPERATING_SYSTEM
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_RESET_REQUIRED
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_TIMEOUT
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_UNINITIALIZED
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_UNKNOWN
 - nvmlDeviceEnumvs, [26](#)
- NVML_ERROR_VGPU_ECC_NOT_SUPPORTED
 - nvmlDeviceEnumvs, [26](#)
- NVML_FAN_FAILED
 - nvmlUnitStructs, [42](#)
- NVML_FAN_NORMAL
 - nvmlUnitStructs, [42](#)
- NVML_FBC_SESSION_TYPE_CUDA
 - nvmlFBCStructs, [50](#)
- NVML_FBC_SESSION_TYPE_HWENC
 - nvmlFBCStructs, [50](#)
- NVML_FBC_SESSION_TYPE_TOSYS
 - nvmlFBCStructs, [50](#)
- NVML_FBC_SESSION_TYPE_UNKNOWN
 - nvmlFBCStructs, [50](#)
- NVML_FBC_SESSION_TYPE_VID
 - nvmlFBCStructs, [50](#)
- NVML_FEATURE_DISABLED
 - nvmlDeviceEnumvs, [23](#)
- NVML_FEATURE_ENABLED
 - nvmlDeviceEnumvs, [23](#)
- NVML_GOM_ALL_ON
 - nvmlDeviceEnumvs, [24](#)
- NVML_GOM_COMPUTE
 - nvmlDeviceEnumvs, [24](#)
- NVML_GOM_LOW_DP
 - nvmlDeviceEnumvs, [24](#)
- NVML_GPU_UTILIZATION_SAMPLES
 - nvmlDeviceStructs, [18](#)
- NVML_GPU_VIRTUALIZATION_MODE_HOST_-
 - VGPU
 - nvmlGridEnums, [28](#)
- NVML_GPU_VIRTUALIZATION_MODE_HOST_-
 - VSGA

- nvmlGridEnums, [28](#)
- NVML_GPU_VIRTUALIZATION_MODE_NONE
 - nvmlGridEnums, [28](#)
- NVML_GPU_VIRTUALIZATION_MODE_-PASSTHROUGH
 - nvmlGridEnums, [28](#)
- NVML_GPU_VIRTUALIZATION_MODE_VGPU
 - nvmlGridEnums, [28](#)
- NVML_GRID_LICENSE_FEATURE_CODE_VGPU
 - nvmlGridEnums, [28](#)
- NVML_GRID_LICENSE_FEATURE_CODE_-VWORKSTATION
 - nvmlGridEnums, [28](#)
- NVML_HOST_VGPU_MODE_NON_SRIOV
 - nvmlGridEnums, [29](#)
- NVML_HOST_VGPU_MODE_SRIOV
 - nvmlGridEnums, [29](#)
- NVML_INFOROM_COUNT
 - nvmlDeviceEnumvs, [24](#)
- NVML_INFOROM_ECC
 - nvmlDeviceEnumvs, [24](#)
- NVML_INFOROM_OEM
 - nvmlDeviceEnumvs, [24](#)
- NVML_INFOROM_POWER
 - nvmlDeviceEnumvs, [24](#)
- NVML_LED_COLOR_AMBER
 - nvmlUnitStructs, [42](#)
- NVML_LED_COLOR_GREEN
 - nvmlUnitStructs, [42](#)
- NVML_MEMORY_CLK_SAMPLES
 - nvmlDeviceStructs, [18](#)
- NVML_MEMORY_ERROR_TYPE_CORRECTED
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_ERROR_TYPE_COUNT
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_ERROR_TYPE_UNCORRECTED
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_LOCATION_CBU
 - nvmlDeviceEnumvs, [25](#)
- NVML_MEMORY_LOCATION_COUNT
 - nvmlDeviceEnumvs, [25](#)
- NVML_MEMORY_LOCATION_DEVICE_MEMORY
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_LOCATION_DRAM
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_LOCATION_L1_CACHE
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_LOCATION_L2_CACHE
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_LOCATION_REGISTER_FILE
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_LOCATION_SRAM
 - nvmlDeviceEnumvs, [25](#)
- NVML_MEMORY_LOCATION_TEXTURE_-MEMORY
 - nvmlDeviceEnumvs, [24](#)
- NVML_MEMORY_LOCATION_TEXTURE_SHM
 - nvmlDeviceEnumvs, [25](#)
- NVML_MEMORY_UTILIZATION_SAMPLES
 - nvmlDeviceStructs, [18](#)
- NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_-BIT_ECC_ERROR
 - nvmlDeviceEnumvs, [25](#)
- NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_-SINGLE_BIT_ECC_ERRORS
 - nvmlDeviceEnumvs, [25](#)
- NVML_PERF_POLICY_BOARD_LIMIT
 - nvmlDeviceStructs, [18](#)
- NVML_PERF_POLICY_LOW_UTILIZATION
 - nvmlDeviceStructs, [18](#)
- NVML_PERF_POLICY_POWER
 - nvmlDeviceStructs, [18](#)
- NVML_PERF_POLICY_RELIABILITY
 - nvmlDeviceStructs, [18](#)
- NVML_PERF_POLICY_SYNC_BOOST
 - nvmlDeviceStructs, [18](#)
- NVML_PERF_POLICY_THERMAL
 - nvmlDeviceStructs, [18](#)
- NVML_PERF_POLICY_TOTAL_APP_CLOCKS
 - nvmlDeviceStructs, [18](#)
- NVML_PERF_POLICY_TOTAL_BASE_CLOCKS
 - nvmlDeviceStructs, [18](#)
- NVML_PROCESSOR_CLK_SAMPLES
 - nvmlDeviceStructs, [18](#)
- NVML_PSTATE_0
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_1
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_10
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_11
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_12
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_13
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_14
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_15
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_2
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_3
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_4
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_5

- nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_6
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_7
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_8
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_9
 - nvmlDeviceEnumvs, [25](#)
- NVML_PSTATE_UNKNOWN
 - nvmlDeviceEnumvs, [25](#)
- NVML_RESTRICTED_API_SET_APPLICATION_-
CLOCKS
 - nvmlDeviceEnumvs, [26](#)
- NVML_RESTRICTED_API_SET_AUTO_-
BOOSTED_CLOCKS
 - nvmlDeviceEnumvs, [26](#)
- NVML_SUCCESS
 - nvmlDeviceEnumvs, [26](#)
- NVML_TEMPERATURE_GPU
 - nvmlDeviceEnumvs, [27](#)
- NVML_TOTAL_POWER_SAMPLES
 - nvmlDeviceStructs, [18](#)
- NVML_VGPU_COMPATIBILITY_LIMIT_GPU
 - nvml, [165](#)
- NVML_VGPU_COMPATIBILITY_LIMIT_GUEST_-
DRIVER
 - nvml, [165](#)
- NVML_VGPU_COMPATIBILITY_LIMIT_HOST_-
DRIVER
 - nvml, [165](#)
- NVML_VGPU_COMPATIBILITY_LIMIT_NONE
 - nvml, [165](#)
- NVML_VGPU_COMPATIBILITY_LIMIT_OTHER
 - nvml, [165](#)
- NVML_VGPU_INSTANCE_GUEST_INFO_STATE_-
INITIALIZED
 - nvmlGridEnums, [29](#)
- NVML_VGPU_INSTANCE_GUEST_INFO_STATE_-
UNINITIALIZED
 - nvmlGridEnums, [29](#)
- NVML_VGPU_VM_COMPATIBILITY_COLD
 - nvml, [165](#)
- NVML_VGPU_VM_COMPATIBILITY_HIBERNATE
 - nvml, [165](#)
- NVML_VGPU_VM_COMPATIBILITY_LIVE
 - nvml, [165](#)
- NVML_VGPU_VM_COMPATIBILITY_NONE
 - nvml, [165](#)
- NVML_VGPU_VM_COMPATIBILITY_SLEEP
 - nvml, [165](#)
- NVML_VGPU_VM_ID_DOMAIN_ID
 - nvmlGridEnums, [29](#)
- NVML_VGPU_VM_ID_UUID
 - nvmlGridEnums, [29](#)
- NVML_VOLATILE_ECC
 - nvmlDeviceEnumvs, [23](#)
- NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE
 - nvmlMultiInstanceGPU, [177](#)
- NVML_CUDA_DRIVER_VERSION_MAJOR
 - nvmlSystemQueries, [57](#)
- NVML_DEVICE_ARCH_KEPLER
 - nvmlVgpuStructs, [31](#)
- NVML_DEVICE_INFOROM_VERSION_BUFFER_-
SIZE
 - nvmlConstants, [55](#)
- NVML_DEVICE_MIG_DISABLE
 - nvmlMultiInstanceGPU, [177](#)
- NVML_DEVICE_MIG_ENABLE
 - nvmlMultiInstanceGPU, [177](#)
- NVML_DEVICE_NAME_BUFFER_SIZE
 - nvmlConstants, [55](#)
- NVML_DEVICE_NAME_V2_BUFFER_SIZE
 - nvmlConstants, [55](#)
- NVML_DEVICE_PART_NUMBER_BUFFER_SIZE
 - nvmlConstants, [55](#)
- NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE
 - nvmlDeviceStructs, [16](#)
- NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE
 - nvmlDeviceStructs, [16](#)
- NVML_DEVICE_PCI_BUS_ID_FMT
 - nvmlDeviceStructs, [16](#)
- NVML_DEVICE_PCI_BUS_ID_FMT_ARGS
 - nvmlDeviceStructs, [16](#)
- NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT
 - nvmlDeviceStructs, [16](#)
- NVML_DEVICE_SERIAL_BUFFER_SIZE
 - nvmlConstants, [55](#)
- NVML_DEVICE_UUID_BUFFER_SIZE
 - nvmlConstants, [55](#)
- NVML_DEVICE_UUID_V2_BUFFER_SIZE
 - nvmlConstants, [55](#)
- NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE
 - nvmlConstants, [55](#)
- NVML_DOUBLE_BIT_ECC
 - nvmlDeviceEnumvs, [21](#)
- NVML_FI_DEV_ECC_CURRENT
 - nvmlFieldValueEnums, [41](#)
- NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID
 - nvmlFieldValueEnums, [41](#)
- NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_-
TX
 - nvmlFieldValueEnums, [41](#)
- NVML_GPU_INSTANCE_PROFILE_1_SLICE
 - nvmlMultiInstanceGPU, [177](#)
- NVML_GRID_LICENSE_BUFFER_SIZE
 - nvmlVgpuConstants, [30](#)
- NVML_MAX_PHYSICAL_BRIDGE

- nvmlDeviceStructs, [17](#)
- NVML_NVLINK_MAX_LINKS
 - nvmlDeviceStructs, [17](#)
- NVML_SINGLE_BIT_ECC
 - nvmlDeviceEnumvs, [21](#)
- NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE
 - nvmlConstants, [56](#)
- NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE
 - nvmlConstants, [56](#)
- NVML_VALUE_NOT_AVAILABLE
 - nvmlDeviceStructs, [17](#)
- NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION
 - nvmlVgpuConstants, [30](#)
- NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION
 - nvmlVgpuConstants, [30](#)
- nvmlAccountingStats
 - nvmlDeviceClearAccountingPids, [45](#)
 - nvmlDeviceGetAccountingBufferSize, [45](#)
 - nvmlDeviceGetAccountingMode, [46](#)
 - nvmlDeviceGetAccountingPids, [46](#)
 - nvmlDeviceGetAccountingStats, [47](#)
 - nvmlDeviceSetAccountingMode, [48](#)
- nvmlAccountingStats_t, [191](#)
- nvmlAffinity
 - nvmlDeviceClearCpuAffinity, [118](#)
 - nvmlDeviceGetCpuAffinity, [118](#)
 - nvmlDeviceGetCpuAffinityWithinScope, [119](#)
 - nvmlDeviceGetMemoryAffinity, [120](#)
 - nvmlDeviceSetCpuAffinity, [120](#)
- nvmlBAR1Memory_t, [193](#)
- nvmlBlacklistDeviceInfo_t, [194](#)
- nvmlBrandType_t
 - nvmlDeviceEnumvs, [22](#)
- nvmlBridgeChipHierarchy_t, [195](#)
- nvmlBridgeChipInfo_t, [196](#)
- nvmlBridgeChipType_t
 - nvmlDeviceStructs, [17](#)
- nvmlClockId_t
 - nvmlDeviceEnumvs, [22](#)
- nvmlClocksThrottleReasonAll
 - nvmlClocksThrottleReasons, [188](#)
- nvmlClocksThrottleReasonApplicationsClocksSetting
 - nvmlClocksThrottleReasons, [188](#)
- nvmlClocksThrottleReasonDisplayClockSetting
 - nvmlClocksThrottleReasons, [188](#)
- nvmlClocksThrottleReasonGpuIdle
 - nvmlClocksThrottleReasons, [188](#)
- nvmlClocksThrottleReasonHwPowerBrakeSlowdown
 - nvmlClocksThrottleReasons, [189](#)
- nvmlClocksThrottleReasonHwSlowdown
 - nvmlClocksThrottleReasons, [189](#)
- nvmlClocksThrottleReasonHwThermalSlowdown
 - nvmlClocksThrottleReasons, [189](#)
- nvmlClocksThrottleReasonNone
 - nvmlClocksThrottleReasons, [189](#)
- NvmlClocksThrottleReasons, [188](#)
- nvmlClocksThrottleReasons
 - nvmlClocksThrottleReasonAll, [188](#)
 - nvmlClocksThrottleReasonApplicationsClocksSetting, [188](#)
 - nvmlClocksThrottleReasonDisplayClockSetting, [188](#)
 - nvmlClocksThrottleReasonGpuIdle, [188](#)
 - nvmlClocksThrottleReasonHwPowerBrakeSlowdown, [189](#)
 - nvmlClocksThrottleReasonHwSlowdown, [189](#)
 - nvmlClocksThrottleReasonHwThermalSlowdown, [189](#)
 - nvmlClocksThrottleReasonNone, [189](#)
 - nvmlClocksThrottleReasonSwPowerCap, [190](#)
 - nvmlClocksThrottleReasonSwThermalSlowdown, [190](#)
 - nvmlClocksThrottleReasonSyncBoost, [190](#)
 - nvmlClocksThrottleReasonUserDefinedClocks, [190](#)
- nvmlClocksThrottleReasonSwPowerCap
 - nvmlClocksThrottleReasons, [190](#)
- nvmlClocksThrottleReasonSwThermalSlowdown
 - nvmlClocksThrottleReasons, [190](#)
- nvmlClocksThrottleReasonSyncBoost
 - nvmlClocksThrottleReasons, [190](#)
- nvmlClocksThrottleReasonUserDefinedClocks
 - nvmlClocksThrottleReasons, [190](#)
- nvmlClockType_t
 - nvmlDeviceEnumvs, [22](#)
- nvmlComputeInstanceDestroy
 - nvmlMultiInstanceGPU, [177](#)
- nvmlComputeInstanceGetInfo
 - nvmlMultiInstanceGPU, [178](#)
- nvmlComputeMode_t
 - nvmlDeviceEnumvs, [22](#)
- nvmlConstants
 - NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE, [55](#)
 - NVML_DEVICE_NAME_BUFFER_SIZE, [55](#)
 - NVML_DEVICE_NAME_V2_BUFFER_SIZE, [55](#)
 - NVML_DEVICE_PART_NUMBER_BUFFER_SIZE, [55](#)
 - NVML_DEVICE_SERIAL_BUFFER_SIZE, [55](#)
 - NVML_DEVICE_UUID_BUFFER_SIZE, [55](#)
 - NVML_DEVICE_UUID_V2_BUFFER_SIZE, [55](#)
 - NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE, [55](#)
 - NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE, [56](#)

- NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE, 56
- nvmlDetachGpuState_t
 - nvmlDrainDefs, 51
- nvmlDeviceClearAccountingPids
 - nvmlAccountingStats, 45
- nvmlDeviceClearCpuAffinity
 - nvmlAffinity, 118
- nvmlDeviceClearEccErrorCounts
 - nvmlDeviceCommands, 123
- nvmlDeviceCommands
 - nvmlDeviceClearEccErrorCounts, 123
 - nvmlDeviceResetGpuLockedClocks, 124
 - nvmlDeviceSetAPIRestriction, 124
 - nvmlDeviceSetApplicationsClocks, 125
 - nvmlDeviceSetComputeMode, 126
 - nvmlDeviceSetDriverModel, 126
 - nvmlDeviceSetEccMode, 127
 - nvmlDeviceSetGpuLockedClocks, 128
 - nvmlDeviceSetGpuOperationMode, 129
 - nvmlDeviceSetPersistenceMode, 129
 - nvmlDeviceSetPowerManagementLimit, 130
- nvmlDeviceCreateGpuInstance
 - nvmlMultiInstanceGPU, 178
- nvmlDeviceDiscoverGpus
 - nvmlZPI, 142
- nvmlDeviceEnumvs
 - NVML_AGGREGATE_ECC, 23
 - NVML_CLOCK_COUNT, 22
 - NVML_CLOCK_GRAPHICS, 22
 - NVML_CLOCK_ID_APP_CLOCK_DEFAULT, 22
 - NVML_CLOCK_ID_APP_CLOCK_TARGET, 22
 - NVML_CLOCK_ID_COUNT, 22
 - NVML_CLOCK_ID_CURRENT, 22
 - NVML_CLOCK_ID_CUSTOMER_BOOST_MAX, 22
 - NVML_CLOCK_MEM, 22
 - NVML_CLOCK_SM, 22
 - NVML_CLOCK_VIDEO, 22
 - NVML_COMPUTEMODE_DEFAULT, 23
 - NVML_COMPUTEMODE_EXCLUSIVE_PROCESS, 23
 - NVML_COMPUTEMODE_EXCLUSIVE_THREAD, 23
 - NVML_COMPUTEMODE_PROHIBITED, 23
 - NVML_DRIVER_WDDM, 23
 - NVML_DRIVER_WDM, 23
 - NVML_ECC_COUNTER_TYPE_COUNT, 23
 - NVML_ERROR_ALREADY_INITIALIZED, 26
 - NVML_ERROR_CORRUPTED_INFOROM, 26
 - NVML_ERROR_DRIVER_NOT_LOADED, 26
 - NVML_ERROR_FUNCTION_NOT_FOUND, 26
 - NVML_ERROR_GPU_IS_LOST, 26
 - NVML_ERROR_IN_USE, 26
 - NVML_ERROR_INSUFFICIENT_POWER, 26
 - NVML_ERROR_INSUFFICIENT_RESOURCES, 26
 - NVML_ERROR_INSUFFICIENT_SIZE, 26
 - NVML_ERROR_INVALID_ARGUMENT, 26
 - NVML_ERROR_IRQ_ISSUE, 26
 - NVML_ERROR_LIB_RM_VERSION_MISMATCH, 26
 - NVML_ERROR_LIBRARY_NOT_FOUND, 26
 - NVML_ERROR_MEMORY, 26
 - NVML_ERROR_NO_DATA, 26
 - NVML_ERROR_NO_PERMISSION, 26
 - NVML_ERROR_NOT_FOUND, 26
 - NVML_ERROR_NOT_SUPPORTED, 26
 - NVML_ERROR_OPERATING_SYSTEM, 26
 - NVML_ERROR_RESET_REQUIRED, 26
 - NVML_ERROR_TIMEOUT, 26
 - NVML_ERROR_UNINITIALIZED, 26
 - NVML_ERROR_UNKNOWN, 26
 - NVML_ERROR_VGPU_ECC_NOT_SUPPORTED, 26
 - NVML_FEATURE_DISABLED, 23
 - NVML_FEATURE_ENABLED, 23
 - NVML_GOM_ALL_ON, 24
 - NVML_GOM_COMPUTE, 24
 - NVML_GOM_LOW_DP, 24
 - NVML_INFOROM_COUNT, 24
 - NVML_INFOROM_ECC, 24
 - NVML_INFOROM_OEM, 24
 - NVML_INFOROM_POWER, 24
 - NVML_MEMORY_ERROR_TYPE_CORRECTED, 24
 - NVML_MEMORY_ERROR_TYPE_COUNT, 24
 - NVML_MEMORY_ERROR_TYPE_UNCORRECTED, 24
 - NVML_MEMORY_LOCATION_CBU, 25
 - NVML_MEMORY_LOCATION_COUNT, 25
 - NVML_MEMORY_LOCATION_DEVICE_MEMORY, 24
 - NVML_MEMORY_LOCATION_DRAM, 24
 - NVML_MEMORY_LOCATION_L1_CACHE, 24
 - NVML_MEMORY_LOCATION_L2_CACHE, 24
 - NVML_MEMORY_LOCATION_REGISTER_FILE, 24
 - NVML_MEMORY_LOCATION_SRAM, 25
 - NVML_MEMORY_LOCATION_TEXTURE_MEMORY, 24
 - NVML_MEMORY_LOCATION_TEXTURE_SHM, 25
 - NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR, 25
 - NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS, 25

- NVML_PSTATE_0, [25](#)
- NVML_PSTATE_1, [25](#)
- NVML_PSTATE_10, [25](#)
- NVML_PSTATE_11, [25](#)
- NVML_PSTATE_12, [25](#)
- NVML_PSTATE_13, [25](#)
- NVML_PSTATE_14, [25](#)
- NVML_PSTATE_15, [25](#)
- NVML_PSTATE_2, [25](#)
- NVML_PSTATE_3, [25](#)
- NVML_PSTATE_4, [25](#)
- NVML_PSTATE_5, [25](#)
- NVML_PSTATE_6, [25](#)
- NVML_PSTATE_7, [25](#)
- NVML_PSTATE_8, [25](#)
- NVML_PSTATE_9, [25](#)
- NVML_PSTATE_UNKNOWN, [25](#)
- NVML_RESTRICTED_API_SET_-
APPLICATION_CLOCKS, [26](#)
- NVML_RESTRICTED_API_SET_AUTO_-
BOOSTED_CLOCKS, [26](#)
- NVML_SUCCESS, [26](#)
- NVML_TEMPERATURE_GPU, [27](#)
- NVML_VOLATILE_ECC, [23](#)
- NVML_DOUBLE_BIT_ECC, [21](#)
- NVML_SINGLE_BIT_ECC, [21](#)
- nvmlBrandType_t, [22](#)
- nvmlClockId_t, [22](#)
- nvmlClockType_t, [22](#)
- nvmlComputeMode_t, [22](#)
- nvmlDriverModel_t, [23](#)
- nvmlEccBitType_t, [22](#)
- nvmlEccCounterType_t, [23](#)
- nvmlEnableState_t, [23](#)
- nvmlGpuOperationMode_t, [23](#)
- nvmlInforomObject_t, [24](#)
- nvmlMemoryErrorType_t, [24](#)
- nvmlMemoryLocation_t, [24](#)
- nvmlPageRetirementCause_t, [25](#)
- nvmlPstates_t, [25](#)
- nvmlRestrictedAPI_t, [25](#)
- nvmlReturn_t, [26](#)
- nvmlTemperatureSensors_t, [26](#)
- nvmlTemperatureThresholds_t, [27](#)
- nvmlDeviceFreezeNvLinkUtilizationCounter
NvLink, [132](#)
- nvmlDeviceGetAccountingBufferSize
nvmlAccountingStats, [45](#)
- nvmlDeviceGetAccountingMode
nvmlAccountingStats, [46](#)
- nvmlDeviceGetAccountingPids
nvmlAccountingStats, [46](#)
- nvmlDeviceGetAccountingStats
nvmlAccountingStats, [47](#)
- nvmlDeviceGetActiveVgpus
nvmlVgpu, [151](#)
- nvmlDeviceGetAPIRestriction
nvmlDeviceQueries, [67](#)
- nvmlDeviceGetApplicationsClock
nvmlDeviceQueries, [67](#)
- nvmlDeviceGetArchitecture
nvmlDeviceQueries, [68](#)
- nvmlDeviceGetAttributes_v2
nvmlDeviceQueries, [68](#)
- nvmlDeviceGetAutoBoostedClocksEnabled
nvmlDeviceQueries, [69](#)
- nvmlDeviceGetBAR1MemoryInfo
nvmlDeviceQueries, [69](#)
- nvmlDeviceGetBoardId
nvmlDeviceQueries, [70](#)
- nvmlDeviceGetBoardPartNumber
nvmlDeviceQueries, [70](#)
- nvmlDeviceGetBrand
nvmlDeviceQueries, [71](#)
- nvmlDeviceGetBridgeChipInfo
nvmlDeviceQueries, [71](#)
- nvmlDeviceGetClock
nvmlDeviceQueries, [71](#)
- nvmlDeviceGetClockInfo
nvmlDeviceQueries, [72](#)
- nvmlDeviceGetComputeInstanceId
nvmlMultiInstanceGPU, [178](#)
- nvmlDeviceGetComputeMode
nvmlDeviceQueries, [72](#)
- nvmlDeviceGetComputeRunningProcesses_v2
nvmlDeviceQueries, [73](#)
- nvmlDeviceGetCount_v2
nvmlDeviceQueries, [74](#)
- nvmlDeviceGetCpuAffinity
nvmlAffinity, [118](#)
- nvmlDeviceGetCpuAffinityWithinScope
nvmlAffinity, [119](#)
- nvmlDeviceGetCreatableVgpus
nvmlVgpu, [151](#)
- nvmlDeviceGetCudaComputeCapability
nvmlDeviceQueries, [74](#)
- nvmlDeviceGetCurrentClocksThrottleReasons
nvmlDeviceQueries, [75](#)
- nvmlDeviceGetCurrPcieLinkGeneration
nvmlDeviceQueries, [75](#)
- nvmlDeviceGetCurrPcieLinkWidth
nvmlDeviceQueries, [76](#)
- nvmlDeviceGetDecoderUtilization
nvmlDeviceQueries, [76](#)
- nvmlDeviceGetDefaultApplicationsClock
nvmlDeviceQueries, [77](#)
- nvmlDeviceGetDetailedEccErrors
nvmlDeviceQueries, [77](#)

- nvmlDeviceGetDeviceHandleFromMigDeviceHandle
 - nvmlMultiInstanceGPU, [179](#)
- nvmlDeviceGetDisplayActive
 - nvmlDeviceQueries, [78](#)
- nvmlDeviceGetDisplayMode
 - nvmlDeviceQueries, [79](#)
- nvmlDeviceGetDriverModel
 - nvmlDeviceQueries, [79](#)
- nvmlDeviceGetEccMode
 - nvmlDeviceQueries, [80](#)
- nvmlDeviceGetEncoderCapacity
 - nvmlDeviceQueries, [80](#)
- nvmlDeviceGetEncoderSessions
 - nvmlDeviceQueries, [81](#)
- nvmlDeviceGetEncoderStats
 - nvmlDeviceQueries, [81](#)
- nvmlDeviceGetEncoderUtilization
 - nvmlDeviceQueries, [82](#)
- nvmlDeviceGetEnforcedPowerLimit
 - nvmlDeviceQueries, [82](#)
- nvmlDeviceGetFanSpeed
 - nvmlDeviceQueries, [83](#)
- nvmlDeviceGetFanSpeed_v2
 - nvmlDeviceQueries, [83](#)
- nvmlDeviceGetFBCSessions
 - nvmlDeviceQueries, [84](#)
- nvmlDeviceGetFBCStats
 - nvmlDeviceQueries, [84](#)
- nvmlDeviceGetFieldValues
 - nvmlFieldValueQueries, [145](#)
- nvmlDeviceGetGpuInstanceById
 - nvmlMultiInstanceGPU, [179](#)
- nvmlDeviceGetGpuInstanceId
 - nvmlMultiInstanceGPU, [180](#)
- nvmlDeviceGetGpuInstancePossiblePlacements
 - nvmlMultiInstanceGPU, [180](#)
- nvmlDeviceGetGpuInstanceProfileInfo
 - nvmlMultiInstanceGPU, [181](#)
- nvmlDeviceGetGpuInstanceRemainingCapacity
 - nvmlMultiInstanceGPU, [181](#)
- nvmlDeviceGetGpuInstances
 - nvmlMultiInstanceGPU, [181](#)
- nvmlDeviceGetGpuOperationMode
 - nvmlDeviceQueries, [85](#)
- nvmlDeviceGetGraphicsRunningProcesses_v2
 - nvmlDeviceQueries, [85](#)
- nvmlDeviceGetGridLicensableFeatures_v3
 - nvmlGridQueries, [147](#)
- nvmlDeviceGetHandleByIndex_v2
 - nvmlDeviceQueries, [86](#)
- nvmlDeviceGetHandleByPciBusId_v2
 - nvmlDeviceQueries, [87](#)
- nvmlDeviceGetHandleBySerial
 - nvmlDeviceQueries, [88](#)
- nvmlDeviceGetHandleByUUID
 - nvmlDeviceQueries, [89](#)
- nvmlDeviceGetHostVgpuMode
 - nvmlGridQueries, [147](#)
- nvmlDeviceGetIndex
 - nvmlDeviceQueries, [89](#)
- nvmlDeviceGetInforomConfigurationChecksum
 - nvmlDeviceQueries, [90](#)
- nvmlDeviceGetInforomImageVersion
 - nvmlDeviceQueries, [90](#)
- nvmlDeviceGetInforomVersion
 - nvmlDeviceQueries, [91](#)
- nvmlDeviceGetMaxClockInfo
 - nvmlDeviceQueries, [92](#)
- nvmlDeviceGetMaxCustomerBoostClock
 - nvmlDeviceQueries, [92](#)
- nvmlDeviceGetMaxMigDeviceCount
 - nvmlMultiInstanceGPU, [182](#)
- nvmlDeviceGetMaxPcieLinkGeneration
 - nvmlDeviceQueries, [93](#)
- nvmlDeviceGetMaxPcieLinkWidth
 - nvmlDeviceQueries, [93](#)
- nvmlDeviceGetMemoryAffinity
 - nvmlAffinity, [120](#)
- nvmlDeviceGetMemoryErrorCounter
 - nvmlDeviceQueries, [94](#)
- nvmlDeviceGetMemoryInfo
 - nvmlDeviceQueries, [94](#)
- nvmlDeviceGetMigDeviceHandleByIndex
 - nvmlMultiInstanceGPU, [182](#)
- nvmlDeviceGetMigMode
 - nvmlMultiInstanceGPU, [183](#)
- nvmlDeviceGetMinorNumber
 - nvmlDeviceQueries, [95](#)
- nvmlDeviceGetMultiGpuBoard
 - nvmlDeviceQueries, [95](#)
- nvmlDeviceGetName
 - nvmlDeviceQueries, [96](#)
- nvmlDeviceGetNvLinkCapability
 - NvLink, [133](#)
- nvmlDeviceGetNvLinkErrorCounter
 - NvLink, [133](#)
- nvmlDeviceGetNvLinkRemotePciInfo_v2
 - NvLink, [133](#)
- nvmlDeviceGetNvLinkState
 - NvLink, [134](#)
- nvmlDeviceGetNvLinkUtilizationControl
 - NvLink, [134](#)
- nvmlDeviceGetNvLinkUtilizationCounter
 - NvLink, [135](#)
- nvmlDeviceGetNvLinkVersion
 - NvLink, [135](#)
- nvmlDeviceGetP2PStatus
 - nvmlDeviceQueries, [96](#)

- nvmlDeviceGetPcieReplayCounter
 - nvmlDeviceQueries, [97](#)
- nvmlDeviceGetPcieThroughput
 - nvmlDeviceQueries, [97](#)
- nvmlDeviceGetPciInfo_v3
 - nvmlDeviceQueries, [98](#)
- nvmlDeviceGetPerformanceState
 - nvmlDeviceQueries, [98](#)
- nvmlDeviceGetPersistenceMode
 - nvmlDeviceQueries, [99](#)
- nvmlDeviceGetPgpuMetadataString
 - nvml, [165](#)
- nvmlDeviceGetPowerManagementDefaultLimit
 - nvmlDeviceQueries, [99](#)
- nvmlDeviceGetPowerManagementLimit
 - nvmlDeviceQueries, [100](#)
- nvmlDeviceGetPowerManagementLimitConstraints
 - nvmlDeviceQueries, [100](#)
- nvmlDeviceGetPowerManagementMode
 - nvmlDeviceQueries, [101](#)
- nvmlDeviceGetPowerState
 - nvmlDeviceQueries, [101](#)
- nvmlDeviceGetPowerUsage
 - nvmlDeviceQueries, [102](#)
- nvmlDeviceGetProcessUtilization
 - nvmlGridQueries, [148](#)
- nvmlDeviceGetRemappedRows
 - nvmlDeviceQueries, [102](#)
- nvmlDeviceGetRetiredPages
 - nvmlDeviceQueries, [103](#)
- nvmlDeviceGetRetiredPages_v2
 - nvmlDeviceQueries, [103](#)
- nvmlDeviceGetRetiredPagesPendingStatus
 - nvmlDeviceQueries, [104](#)
- nvmlDeviceGetRowRemapperHistogram
 - nvmlDeviceQueries, [105](#)
- nvmlDeviceGetSamples
 - nvmlDeviceQueries, [105](#)
- nvmlDeviceGetSerial
 - nvmlDeviceQueries, [106](#)
- nvmlDeviceGetSupportedClocksThrottleReasons
 - nvmlDeviceQueries, [106](#)
- nvmlDeviceGetSupportedEventTypes
 - nvmlEvents, [138](#)
- nvmlDeviceGetSupportedGraphicsClocks
 - nvmlDeviceQueries, [107](#)
- nvmlDeviceGetSupportedMemoryClocks
 - nvmlDeviceQueries, [108](#)
- nvmlDeviceGetSupportedVgpu
 - nvmlVgpu, [152](#)
- nvmlDeviceGetTemperature
 - nvmlDeviceQueries, [108](#)
- nvmlDeviceGetTemperatureThreshold
 - nvmlDeviceQueries, [109](#)
- nvmlDeviceGetTopologyCommonAncestor
 - nvmlDeviceQueries, [109](#)
- nvmlDeviceGetTopologyNearestGpus
 - nvmlDeviceQueries, [109](#)
- nvmlDeviceGetTotalEccErrors
 - nvmlDeviceQueries, [110](#)
- nvmlDeviceGetTotalEnergyConsumption
 - nvmlDeviceQueries, [111](#)
- nvmlDeviceGetUtilizationRates
 - nvmlDeviceQueries, [111](#)
- nvmlDeviceGetUUID
 - nvmlDeviceQueries, [112](#)
- nvmlDeviceGetVbiosVersion
 - nvmlDeviceQueries, [112](#)
- nvmlDeviceGetVgpuMetadata
 - nvml, [165](#)
- nvmlDeviceGetVgpuProcessUtilization
 - nvmlUtil, [169](#)
- nvmlDeviceGetVgpuUtilization
 - nvmlUtil, [170](#)
- nvmlDeviceGetViolationStatus
 - nvmlDeviceQueries, [113](#)
- nvmlDeviceGetVirtualizationMode
 - nvmlGridQueries, [148](#)
- nvmlDeviceIsMigDeviceHandle
 - nvmlMultiInstanceGPU, [183](#)
- nvmlDeviceModifyDrainState
 - nvmlZPI, [142](#)
- nvmlDeviceOnSameBoard
 - nvmlDeviceQueries, [113](#)
- nvmlDeviceQueries
 - nvmlDeviceGetAPIRestriction, [67](#)
 - nvmlDeviceGetApplicationsClock, [67](#)
 - nvmlDeviceGetArchitecture, [68](#)
 - nvmlDeviceGetAttributes_v2, [68](#)
 - nvmlDeviceGetAutoBoostedClocksEnabled, [69](#)
 - nvmlDeviceGetBAR1MemoryInfo, [69](#)
 - nvmlDeviceGetBoardId, [70](#)
 - nvmlDeviceGetBoardPartNumber, [70](#)
 - nvmlDeviceGetBrand, [71](#)
 - nvmlDeviceGetBridgeChipInfo, [71](#)
 - nvmlDeviceGetClock, [71](#)
 - nvmlDeviceGetClockInfo, [72](#)
 - nvmlDeviceGetComputeMode, [72](#)
 - nvmlDeviceGetComputeRunningProcesses_v2, [73](#)
 - nvmlDeviceGetCount_v2, [74](#)
 - nvmlDeviceGetCudaComputeCapability, [74](#)
 - nvmlDeviceGetCurrentClocksThrottleReasons, [75](#)
 - nvmlDeviceGetCurrPcieLinkGeneration, [75](#)
 - nvmlDeviceGetCurrPcieLinkWidth, [76](#)
 - nvmlDeviceGetDecoderUtilization, [76](#)
 - nvmlDeviceGetDefaultApplicationsClock, [77](#)
 - nvmlDeviceGetDetailedEccErrors, [77](#)
 - nvmlDeviceGetDisplayActive, [78](#)

- [nvmlDeviceGetDisplayMode, 79](#)
- [nvmlDeviceGetDriverModel, 79](#)
- [nvmlDeviceGetEccMode, 80](#)
- [nvmlDeviceGetEncoderCapacity, 80](#)
- [nvmlDeviceGetEncoderSessions, 81](#)
- [nvmlDeviceGetEncoderStats, 81](#)
- [nvmlDeviceGetEncoderUtilization, 82](#)
- [nvmlDeviceGetEnforcedPowerLimit, 82](#)
- [nvmlDeviceGetFanSpeed, 83](#)
- [nvmlDeviceGetFanSpeed_v2, 83](#)
- [nvmlDeviceGetFBCSessions, 84](#)
- [nvmlDeviceGetFBCStats, 84](#)
- [nvmlDeviceGetGpuOperationMode, 85](#)
- [nvmlDeviceGetGraphicsRunningProcesses_v2, 85](#)
- [nvmlDeviceGetHandleByIndex_v2, 86](#)
- [nvmlDeviceGetHandleByPciBusId_v2, 87](#)
- [nvmlDeviceGetHandleBySerial, 88](#)
- [nvmlDeviceGetHandleByUUID, 89](#)
- [nvmlDeviceGetIndex, 89](#)
- [nvmlDeviceGetInforomConfigurationChecksum, 90](#)
- [nvmlDeviceGetInforomImageVersion, 90](#)
- [nvmlDeviceGetInforomVersion, 91](#)
- [nvmlDeviceGetMaxClockInfo, 92](#)
- [nvmlDeviceGetMaxCustomerBoostClock, 92](#)
- [nvmlDeviceGetMaxPcieLinkGeneration, 93](#)
- [nvmlDeviceGetMaxPcieLinkWidth, 93](#)
- [nvmlDeviceGetMemoryErrorCounter, 94](#)
- [nvmlDeviceGetMemoryInfo, 94](#)
- [nvmlDeviceGetMinorNumber, 95](#)
- [nvmlDeviceGetMultiGpuBoard, 95](#)
- [nvmlDeviceGetName, 96](#)
- [nvmlDeviceGetP2PStatus, 96](#)
- [nvmlDeviceGetPcieReplayCounter, 97](#)
- [nvmlDeviceGetPcieThroughput, 97](#)
- [nvmlDeviceGetPciInfo_v3, 98](#)
- [nvmlDeviceGetPerformanceState, 98](#)
- [nvmlDeviceGetPersistenceMode, 99](#)
- [nvmlDeviceGetPowerManagementDefaultLimit, 99](#)
- [nvmlDeviceGetPowerManagementLimit, 100](#)
- [nvmlDeviceGetPowerManagementLimitConstraints, 100](#)
- [nvmlDeviceGetPowerManagementMode, 101](#)
- [nvmlDeviceGetPowerState, 101](#)
- [nvmlDeviceGetPowerUsage, 102](#)
- [nvmlDeviceGetRemappedRows, 102](#)
- [nvmlDeviceGetRetiredPages, 103](#)
- [nvmlDeviceGetRetiredPages_v2, 103](#)
- [nvmlDeviceGetRetiredPagesPendingStatus, 104](#)
- [nvmlDeviceGetRowRemapperHistogram, 105](#)
- [nvmlDeviceGetSamples, 105](#)
- [nvmlDeviceGetSerial, 106](#)
- [nvmlDeviceGetSupportedClocksThrottleReasons, 106](#)
- [nvmlDeviceGetSupportedGraphicsClocks, 107](#)
- [nvmlDeviceGetSupportedMemoryClocks, 108](#)
- [nvmlDeviceGetTemperature, 108](#)
- [nvmlDeviceGetTemperatureThreshold, 109](#)
- [nvmlDeviceGetTopologyCommonAncestor, 109](#)
- [nvmlDeviceGetTopologyNearestGpus, 109](#)
- [nvmlDeviceGetTotalEccErrors, 110](#)
- [nvmlDeviceGetTotalEnergyConsumption, 111](#)
- [nvmlDeviceGetUtilizationRates, 111](#)
- [nvmlDeviceGetUUID, 112](#)
- [nvmlDeviceGetVbiosVersion, 112](#)
- [nvmlDeviceGetViolationStatus, 113](#)
- [nvmlDeviceOnSameBoard, 113](#)
- [nvmlDeviceResetApplicationsClocks, 114](#)
- [nvmlDeviceSetAutoBoostedClocksEnabled, 114](#)
- [nvmlDeviceSetDefaultAutoBoostedClocksEnabled, 115](#)
- [nvmlDeviceValidateInforom, 115](#)
- [nvmlSystemGetTopologyGpuSet, 116](#)
- [nvmlVgpuInstanceGetMdevUUID, 116](#)
- [nvmlDeviceQueryDrainState](#)
 - [nvmlZPI, 143](#)
- [nvmlDeviceRegisterEvents](#)
 - [nvmlEvents, 139](#)
- [nvmlDeviceRemoveGpu_v2](#)
 - [nvmlZPI, 143](#)
- [nvmlDeviceResetApplicationsClocks](#)
 - [nvmlDeviceQueries, 114](#)
- [nvmlDeviceResetGpuLockedClocks](#)
 - [nvmlDeviceCommands, 124](#)
- [nvmlDeviceResetNvLinkErrorCounters](#)
 - [NvLink, 136](#)
- [nvmlDeviceResetNvLinkUtilizationCounter](#)
 - [NvLink, 136](#)
- [nvmlDeviceSetAccountingMode](#)
 - [nvmlAccountingStats, 48](#)
- [nvmlDeviceSetAPIRestriction](#)
 - [nvmlDeviceCommands, 124](#)
- [nvmlDeviceSetApplicationsClocks](#)
 - [nvmlDeviceCommands, 125](#)
- [nvmlDeviceSetAutoBoostedClocksEnabled](#)
 - [nvmlDeviceQueries, 114](#)
- [nvmlDeviceSetComputeMode](#)
 - [nvmlDeviceCommands, 126](#)
- [nvmlDeviceSetCpuAffinity](#)
 - [nvmlAffinity, 120](#)
- [nvmlDeviceSetDefaultAutoBoostedClocksEnabled](#)
 - [nvmlDeviceQueries, 115](#)
- [nvmlDeviceSetDriverModel](#)
 - [nvmlDeviceCommands, 126](#)
- [nvmlDeviceSetEccMode](#)
 - [nvmlDeviceCommands, 127](#)
- [nvmlDeviceSetGpuLockedClocks](#)
 - [nvmlDeviceCommands, 128](#)
- [nvmlDeviceSetGpuOperationMode](#)

- nvmlDeviceCommands, [129](#)
- nvmlDeviceSetMigMode
 - nvmlMultiInstanceGPU, [184](#)
- nvmlDeviceSetNvLinkUtilizationControl
 - NvLink, [137](#)
- nvmlDeviceSetPersistenceMode
 - nvmlDeviceCommands, [129](#)
- nvmlDeviceSetPowerManagementLimit
 - nvmlDeviceCommands, [130](#)
- nvmlDeviceSetVirtualizationMode
 - nvmlGridQueries, [149](#)
- nvmlDeviceStructs
 - NVML_DEC_UTILIZATION_SAMPLES, [18](#)
 - NVML_ENC_UTILIZATION_SAMPLES, [18](#)
 - NVML_GPU_UTILIZATION_SAMPLES, [18](#)
 - NVML_MEMORY_CLK_SAMPLES, [18](#)
 - NVML_MEMORY_UTILIZATION_SAMPLES, [18](#)
 - NVML_PERF_POLICY_BOARD_LIMIT, [18](#)
 - NVML_PERF_POLICY_LOW_UTILIZATION, [18](#)
 - NVML_PERF_POLICY_POWER, [18](#)
 - NVML_PERF_POLICY_RELIABILITY, [18](#)
 - NVML_PERF_POLICY_SYNC_BOOST, [18](#)
 - NVML_PERF_POLICY_THERMAL, [18](#)
 - NVML_PERF_POLICY_TOTAL_APP_CLOCKS, [18](#)
 - NVML_PERF_POLICY_TOTAL_BASE_CLOCKS, [18](#)
 - NVML_PROCESSOR_CLK_SAMPLES, [18](#)
 - NVML_TOTAL_POWER_SAMPLES, [18](#)
 - NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE, [16](#)
 - NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE, [16](#)
 - NVML_DEVICE_PCI_BUS_ID_FMT, [16](#)
 - NVML_DEVICE_PCI_BUS_ID_FMT_ARGS, [16](#)
 - NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT, [16](#)
 - NVML_MAX_PHYSICAL_BRIDGE, [17](#)
 - NVML_NVLINK_MAX_LINKS, [17](#)
 - NVML_VALUE_NOT_AVAILABLE, [17](#)
 - nvmlBridgeChipType_t, [17](#)
 - nvmlGpuTopologyLevel_t, [17](#)
 - nvmlNvLinkCapability_t, [17](#)
 - nvmlNvLinkErrorCounter_t, [17](#)
 - nvmlNvLinkUtilizationCountPktTypes_t, [17](#)
 - nvmlNvLinkUtilizationCountUnits_t, [17](#)
 - nvmlPcieUtilCounter_t, [17](#)
 - nvmlPerfPolicyType_t, [18](#)
 - nvmlSamplingType_t, [18](#)
 - nvmlValueType_t, [18](#)
- nvmlDeviceValidateInforom
 - nvmlDeviceQueries, [115](#)
- nvmlDrainDefs
 - nvmlDetachGpuState_t, [51](#)
 - nvmlPcieLinkState_t, [51](#)
- nvmlDriverModel_t
 - nvmlDeviceEnumvs, [23](#)
- nvmlEccBitType_t
 - nvmlDeviceEnumvs, [22](#)
- nvmlEccCounterType_t
 - nvmlDeviceEnumvs, [23](#)
- nvmlEccErrorCounts_t, [197](#)
- nvmlEnableState_t
 - nvmlDeviceEnumvs, [23](#)
- nvmlEncoderSessionInfo_t, [198](#)
- nvmlEncoderStructs
 - NVML_ENCODER_QUERY_H264, [49](#)
 - NVML_ENCODER_QUERY_HEVC, [49](#)
- nvmlEncoderType_t, [49](#)
- nvmlEncoderType_t
 - nvmlEncoderStructs, [49](#)
- nvmlErrorReporting
 - nvmlErrorString, [54](#)
- nvmlErrorString
 - nvmlErrorReporting, [54](#)
- nvmlEventData_t, [199](#)
- nvmlEvents
 - nvmlDeviceGetSupportedEventTypes, [138](#)
 - nvmlDeviceRegisterEvents, [139](#)
 - nvmlEventSet_t, [138](#)
 - nvmlEventSetCreate, [139](#)
 - nvmlEventSetFree, [140](#)
 - nvmlEventSetWait_v2, [140](#)
- nvmlEventSet_t
 - nvmlEvents, [138](#)
- nvmlEventSetCreate
 - nvmlEvents, [139](#)
- nvmlEventSetFree
 - nvmlEvents, [140](#)
- nvmlEventSetWait_v2
 - nvmlEvents, [140](#)
- nvmlEventType
 - nvmlEventTypeClock, [43](#)
 - nvmlEventTypeDoubleBitEccError, [43](#)
 - nvmlEventTypePState, [43](#)
 - nvmlEventTypeSingleBitEccError, [44](#)
- nvmlEventTypeClock
 - nvmlEventType, [43](#)
- nvmlEventTypeDoubleBitEccError
 - nvmlEventType, [43](#)
- nvmlEventTypePState
 - nvmlEventType, [43](#)
- nvmlEventTypeSingleBitEccError
 - nvmlEventType, [44](#)
- nvmlFanState_t
 - nvmlUnitStructs, [42](#)
- nvmlFBCSessionInfo_t, [200](#)

- nvmlFBCSessionType_t
 - nvmlFBCStructs, 50
- nvmlFBCStats_t, 201
- nvmlFBCStructs
 - NVML_FBC_SESSION_TYPE_CUDA, 50
 - NVML_FBC_SESSION_TYPE_HWENC, 50
 - NVML_FBC_SESSION_TYPE_TOSYS, 50
 - NVML_FBC_SESSION_TYPE_UNKNOWN, 50
 - NVML_FBC_SESSION_TYPE_VID, 50
 - nvmlFBCSessionType_t, 50
- nvmlFieldValue_t, 202
- nvmlFieldValueEnums
 - NVML_FI_DEV_ECC_CURRENT, 41
 - NVML_FI_DEV_NVLINK_REMOTE_NVLINK_ID, 41
 - NVML_FI_DEV_NVLINK_THROUGHPUT_DATA_TX, 41
- nvmlFieldValueQueries
 - nvmlDeviceGetFieldValues, 145
- nvmlGetBlacklistDeviceCount
 - nvmlGpuBlacklistQueries, 174
- nvmlGetBlacklistDeviceInfoByIndex
 - nvmlGpuBlacklistQueries, 174
- nvmlGetVgpuCompatibility
 - nvml, 166
- nvmlGetVgpuVersion
 - nvml, 166
- nvmlGpuBlacklistQueries
 - nvmlGetBlacklistDeviceCount, 174
 - nvmlGetBlacklistDeviceInfoByIndex, 174
- nvmlGpuInstanceCreateComputeInstance
 - nvmlMultiInstanceGPU, 184
- nvmlGpuInstanceDestroy
 - nvmlMultiInstanceGPU, 185
- nvmlGpuInstanceGetComputeInstanceId
 - nvmlMultiInstanceGPU, 185
- nvmlGpuInstanceGetComputeInstanceProfileInfo
 - nvmlMultiInstanceGPU, 186
- nvmlGpuInstanceGetComputeInstanceRemainingCapacity
 - nvmlMultiInstanceGPU, 186
- nvmlGpuInstanceGetComputeInstances
 - nvmlMultiInstanceGPU, 186
- nvmlGpuInstanceGetInfo
 - nvmlMultiInstanceGPU, 187
- nvmlGpuOperationMode_t
 - nvmlDeviceEnumvs, 23
- nvmlGpuTopologyLevel_t
 - nvmlDeviceStructs, 17
- nvmlGpuVirtualizationMode_t
 - nvmlGridEnums, 28
- nvmlGridEnums
 - NVML_GPU_VIRTUALIZATION_MODE_HOST_VGPU, 28
 - NVML_GPU_VIRTUALIZATION_MODE_NONE, 28
 - NVML_GPU_VIRTUALIZATION_MODE_PASSTHROUGH, 28
 - NVML_GPU_VIRTUALIZATION_MODE_VGPU, 28
 - NVML_GRID_LICENSE_FEATURE_CODE_VGPU, 28
 - NVML_GRID_LICENSE_FEATURE_CODE_VWORKSTATION, 28
 - NVML_HOST_VGPU_MODE_NON_SRIOV, 29
 - NVML_HOST_VGPU_MODE_SRIOV, 29
 - NVML_VGPU_INSTANCE_GUEST_INFO_STATE_INITIALIZED, 29
 - NVML_VGPU_INSTANCE_GUEST_INFO_STATE_UNINITIALIZED, 29
 - NVML_VGPU_VM_ID_DOMAIN_ID, 29
 - NVML_VGPU_VM_ID_UUID, 29
 - nvmlGpuVirtualizationMode_t, 28
 - nvmlGridLicenseFeatureCode_t, 28
 - nvmlHostVgpuMode_t, 28
 - nvmlVgpuGuestInfoState_t, 29
 - nvmlVgpuVmIdType_t, 29
- nvmlGridLicensableFeature_t, 203
- nvmlGridLicensableFeatures_t, 204
- nvmlGridLicenseFeatureCode_t
 - nvmlGridEnums, 28
- nvmlGridQueries
 - nvmlDeviceGetGridLicensableFeatures_v3, 147
 - nvmlDeviceGetHostVgpuMode, 147
 - nvmlDeviceGetProcessUtilization, 148
 - nvmlDeviceGetVirtualizationMode, 148
 - nvmlDeviceSetVirtualizationMode, 149
- nvmlHostVgpuMode_t
 - nvmlGridEnums, 28
- nvmlHwbcEntry_t, 205
- nvmlInforomObject_t
 - nvmlDeviceEnumvs, 24
- nvmlInit_v2
 - nvmlInitializationAndCleanup, 52
- nvmlInitializationAndCleanup
 - nvmlInit_v2, 52
 - nvmlInitWithFlags, 53
 - nvmlShutdown, 53
- nvmlInitWithFlags
 - nvmlInitializationAndCleanup, 53
- nvmlLedColor_t
 - nvmlUnitStructs, 42
- nvmlLedState_t, 206
- nvmlMemory_t, 207
- nvmlMemoryErrorType_t
 - nvmlDeviceEnumvs, 24

- nvmlMemoryLocation_t
 - nvmlDeviceEnumvs, [24](#)
- nvmlMultiInstanceGPU
 - NVML_COMPUTE_INSTANCE_PROFILE_1_SLICE, [177](#)
 - NVML_DEVICE_MIG_DISABLE, [177](#)
 - NVML_DEVICE_MIG_ENABLE, [177](#)
 - NVML_GPU_INSTANCE_PROFILE_1_SLICE, [177](#)
 - nvmlComputeInstanceDestroy, [177](#)
 - nvmlComputeInstanceGetInfo, [178](#)
 - nvmlDeviceCreateGpuInstance, [178](#)
 - nvmlDeviceGetComputeInstanceId, [178](#)
 - nvmlDeviceGetDeviceHandleFromMigDeviceHandle, [179](#)
 - nvmlDeviceGetGpuInstanceById, [179](#)
 - nvmlDeviceGetGpuInstanceId, [180](#)
 - nvmlDeviceGetGpuInstancePossiblePlacements, [180](#)
 - nvmlDeviceGetGpuInstanceProfileInfo, [181](#)
 - nvmlDeviceGetGpuInstanceRemainingCapacity, [181](#)
 - nvmlDeviceGetGpuInstances, [181](#)
 - nvmlDeviceGetMaxMigDeviceCount, [182](#)
 - nvmlDeviceGetMigDeviceHandleByIndex, [182](#)
 - nvmlDeviceGetMigMode, [183](#)
 - nvmlDeviceIsMigDeviceHandle, [183](#)
 - nvmlDeviceSetMigMode, [184](#)
 - nvmlGpuInstanceCreateComputeInstance, [184](#)
 - nvmlGpuInstanceDestroy, [185](#)
 - nvmlGpuInstanceGetComputeInstanceById, [185](#)
 - nvmlGpuInstanceGetComputeInstanceProfileInfo, [186](#)
 - nvmlGpuInstanceGetComputeInstanceRemainingCapacity, [186](#)
 - nvmlGpuInstanceGetComputeInstances, [186](#)
 - nvmlGpuInstanceGetInfo, [187](#)
- nvmlNvLinkCapability_t
 - nvmlDeviceStructs, [17](#)
- nvmlNvLinkErrorCounter_t
 - nvmlDeviceStructs, [17](#)
- nvmlNvLinkUtilizationControl_t, [208](#)
- nvmlNvLinkUtilizationCountPktTypes_t
 - nvmlDeviceStructs, [17](#)
- nvmlNvLinkUtilizationCountUnits_t
 - nvmlDeviceStructs, [17](#)
- nvmlPageRetirementCause_t
 - nvmlDeviceEnumvs, [25](#)
- nvmlPcieLinkState_t
 - nvmlDrainDefs, [51](#)
- nvmlPcieUtilCounter_t
 - nvmlDeviceStructs, [17](#)
- nvmlPciInfo_t, [209](#)
- nvmlPerfPolicyType_t
 - nvmlDeviceStructs, [18](#)
- nvmlProcessInfo_t, [210](#)
- nvmlProcessUtilizationSample_t, [211](#)
- nvmlPstates_t
 - nvmlDeviceEnumvs, [25](#)
- nvmlPSUInfo_t, [212](#)
- nvmlRestrictedAPI_t
 - nvmlDeviceEnumvs, [25](#)
- nvmlReturn_t
 - nvmlDeviceEnumvs, [26](#)
- nvmlRowRemapperHistogramValues_t, [213](#)
- nvmlSample_t, [214](#)
- nvmlSamplingType_t
 - nvmlDeviceStructs, [18](#)
- nvmlSetVgpuVersion
 - nvml, [167](#)
- nvmlShutdown
 - nvmlInitializationAndCleanup, [53](#)
- nvmlSystemGetCudaDriverVersion
 - nvmlSystemQueries, [57](#)
- nvmlSystemGetCudaDriverVersion_v2
 - nvmlSystemQueries, [57](#)
- nvmlSystemGetDriverVersion
 - nvmlSystemQueries, [58](#)
- nvmlSystemGetHicVersion
 - nvmlUnitQueries, [60](#)
- nvmlSystemGetNVMLVersion
 - nvmlSystemQueries, [58](#)
- nvmlSystemGetProcessName
 - nvmlSystemQueries, [58](#)
- nvmlSystemGetTopologyGpuSet
 - nvmlDeviceQueries, [116](#)
- nvmlSystemQueries
 - NVML_CUDA_DRIVER_VERSION_MAJOR, [57](#)
 - nvmlSystemGetCudaDriverVersion, [57](#)
 - nvmlSystemGetCudaDriverVersion_v2, [57](#)
 - nvmlSystemGetDriverVersion, [58](#)
 - nvmlSystemGetNVMLVersion, [58](#)
 - nvmlSystemGetProcessName, [58](#)
- nvmlTemperatureSensors_t
 - nvmlDeviceEnumvs, [26](#)
- nvmlTemperatureThresholds_t
 - nvmlDeviceEnumvs, [27](#)
- nvmlUnitCommands
 - nvmlUnitSetLedState, [122](#)
- nvmlUnitFanInfo_t, [215](#)
- nvmlUnitFanSpeeds_t, [216](#)
- nvmlUnitGetCount
 - nvmlUnitQueries, [60](#)
- nvmlUnitGetDevices
 - nvmlUnitQueries, [61](#)
- nvmlUnitGetFanSpeedInfo
 - nvmlUnitQueries, [61](#)
- nvmlUnitGetHandleByIndex

- nvmlUnitQueries, 61
- nvmlUnitGetLedState
 - nvmlUnitQueries, 62
- nvmlUnitGetPsuInfo
 - nvmlUnitQueries, 62
- nvmlUnitGetTemperature
 - nvmlUnitQueries, 63
- nvmlUnitGetUnitInfo
 - nvmlUnitQueries, 63
- nvmlUnitInfo_t, 217
- nvmlUnitQueries
 - nvmlSystemGetHicVersion, 60
 - nvmlUnitGetCount, 60
 - nvmlUnitGetDevices, 61
 - nvmlUnitGetFanSpeedInfo, 61
 - nvmlUnitGetHandleByIndex, 61
 - nvmlUnitGetLedState, 62
 - nvmlUnitGetPsuInfo, 62
 - nvmlUnitGetTemperature, 63
 - nvmlUnitGetUnitInfo, 63
- nvmlUnitSetLedState
 - nvmlUnitCommands, 122
- nvmlUnitStructs
 - NVML_FAN_FAILED, 42
 - NVML_FAN_NORMAL, 42
 - NVML_LED_COLOR_AMBER, 42
 - NVML_LED_COLOR_GREEN, 42
 - nvmlFanState_t, 42
 - nvmlLedColor_t, 42
- nvmlUtil
 - nvmlDeviceGetVgpuProcessUtilization, 169
 - nvmlDeviceGetVgpuUtilization, 170
 - nvmlVgpuInstanceClearAccountingPids, 171
 - nvmlVgpuInstanceGetAccountingMode, 171
 - nvmlVgpuInstanceGetAccountingPids, 172
 - nvmlVgpuInstanceGetAccountingStats, 173
- nvmlUtilization_t, 218
- nvmlValue_t, 219
- nvmlValueType_t
 - nvmlDeviceStructs, 18
- nvmlVgpu
 - nvmlDeviceGetActiveVgpus, 151
 - nvmlDeviceGetCreatableVgpus, 151
 - nvmlDeviceGetSupportedVgpus, 152
 - nvmlVgpuInstanceGetEccMode, 152
 - nvmlVgpuInstanceGetEncoderCapacity, 153
 - nvmlVgpuInstanceGetEncoderSessions, 153
 - nvmlVgpuInstanceGetEncoderStats, 154
 - nvmlVgpuInstanceGetFBCSessions, 154
 - nvmlVgpuInstanceGetFBCStats, 155
 - nvmlVgpuInstanceGetFbUsage, 155
 - nvmlVgpuInstanceGetFrameRateLimit, 156
 - nvmlVgpuInstanceGetLicenseStatus, 156
 - nvmlVgpuInstanceGetType, 157
 - nvmlVgpuInstanceGetUUID, 157
 - nvmlVgpuInstanceGetVmDriverVersion, 158
 - nvmlVgpuInstanceGetVmID, 158
 - nvmlVgpuInstanceSetEncoderCapacity, 159
 - nvmlVgpuTypeGetClass, 159
 - nvmlVgpuTypeGetDeviceID, 160
 - nvmlVgpuTypeGetFramebufferSize, 160
 - nvmlVgpuTypeGetFrameRateLimit, 160
 - nvmlVgpuTypeGetLicense, 161
 - nvmlVgpuTypeGetMaxInstances, 161
 - nvmlVgpuTypeGetMaxInstancesPerVm, 162
 - nvmlVgpuTypeGetName, 162
 - nvmlVgpuTypeGetNumDisplayHeads, 163
 - nvmlVgpuTypeGetResolution, 163
- nvmlVgpuConstants
 - NVML_GRID_LICENSE_BUFFER_SIZE, 30
 - NVML_VGPU_PGPU_VIRTUALIZATION_CAP_MIGRATION, 30
 - NVML_VGPU_VIRTUALIZATION_CAP_MIGRATION, 30
- nvmlVgpuGuestInfoState_t
 - nvmlGridEnums, 29
- nvmlVgpuInstanceClearAccountingPids
 - nvmlUtil, 171
- nvmlVgpuInstanceGetAccountingMode
 - nvmlUtil, 171
- nvmlVgpuInstanceGetAccountingPids
 - nvmlUtil, 172
- nvmlVgpuInstanceGetAccountingStats
 - nvmlUtil, 173
- nvmlVgpuInstanceGetEccMode
 - nvmlVgpu, 152
- nvmlVgpuInstanceGetEncoderCapacity
 - nvmlVgpu, 153
- nvmlVgpuInstanceGetEncoderSessions
 - nvmlVgpu, 153
- nvmlVgpuInstanceGetEncoderStats
 - nvmlVgpu, 154
- nvmlVgpuInstanceGetFBCSessions
 - nvmlVgpu, 154
- nvmlVgpuInstanceGetFBCStats
 - nvmlVgpu, 155
- nvmlVgpuInstanceGetFbUsage
 - nvmlVgpu, 155
- nvmlVgpuInstanceGetFrameRateLimit
 - nvmlVgpu, 156
- nvmlVgpuInstanceGetLicenseStatus
 - nvmlVgpu, 156
- nvmlVgpuInstanceGetMdevUUID
 - nvmlDeviceQueries, 116
- nvmlVgpuInstanceGetMetadata
 - nvml, 168
- nvmlVgpuInstanceGetType
 - nvmlVgpu, 157

- nvmlVgpuInstanceGetUUID
 - nvmlVgpu, [157](#)
- nvmlVgpuInstanceGetVmDriverVersion
 - nvmlVgpu, [158](#)
- nvmlVgpuInstanceGetVmID
 - nvmlVgpu, [158](#)
- nvmlVgpuInstanceSetEncoderCapacity
 - nvmlVgpu, [159](#)
- nvmlVgpuInstanceUtilizationSample_t, [220](#)
- nvmlVgpuMetadata_t, [221](#)
- nvmlVgpuPgpuCompatibility_t, [222](#)
- nvmlVgpuPgpuCompatibilityLimitCode_t
 - nvml, [165](#)
- nvmlVgpuPgpuMetadata_t, [223](#)
- nvmlVgpuProcessUtilizationSample_t, [224](#)
- nvmlVgpuStructs
 - NVML_DEVICE_ARCH_KEPLER, [31](#)
- nvmlVgpuTypeGetClass
 - nvmlVgpu, [159](#)
- nvmlVgpuTypeGetDeviceID
 - nvmlVgpu, [160](#)
- nvmlVgpuTypeGetFramebufferSize
 - nvmlVgpu, [160](#)
- nvmlVgpuTypeGetFrameRateLimit
 - nvmlVgpu, [160](#)
- nvmlVgpuTypeGetLicense
 - nvmlVgpu, [161](#)
- nvmlVgpuTypeGetMaxInstances
 - nvmlVgpu, [161](#)
- nvmlVgpuTypeGetMaxInstancesPerVm
 - nvmlVgpu, [162](#)
- nvmlVgpuTypeGetName
 - nvmlVgpu, [162](#)
- nvmlVgpuTypeGetNumDisplayHeads
 - nvmlVgpu, [163](#)
- nvmlVgpuTypeGetResolution
 - nvmlVgpu, [163](#)
- nvmlVgpuVersion_t, [225](#)
- nvmlVgpuVmCompatibility_t
 - nvml, [165](#)
- nvmlVgpuVmIdType_t
 - nvmlGridEnums, [29](#)
- nvmlViolationTime_t, [226](#)
- nvmlZPI
 - nvmlDeviceDiscoverGpus, [142](#)
 - nvmlDeviceModifyDrainState, [142](#)
 - nvmlDeviceQueryDrainState, [143](#)
 - nvmlDeviceRemoveGpu_v2, [143](#)

System Queries, [57](#)

Unit Commands, [122](#)

Unit Queries, [60](#)

Unit Structs, [42](#)

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, the NVIDIA logo, GeForce, Tesla, and Quadro are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2007-2016 NVIDIA Corporation. All rights reserved.