



Denglin HammingTM V2

dIML API

DL-DG/SW-037A-02

2023-8-17

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。

本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼5号1101室，江苏，中国

<http://www.denglin.ai>

Email : support@denglin.ai

Change History

Version	Change description
02	Add function <code>Function nvmlDeviceGetArchitecture</code> .
01	Initial version.

Contents

LIBRARY API

1 Class Hierarchy

2 Full API

2.1 Classes and Structs

Struct nvmlClusterRemap_st
 Struct nvmlEventData_st
 Struct nvmlMemory_st
 Struct nvmlPciInfo_st
 Struct nvmlProcessInfo_st
 Struct nvmlProcessUtilizationSample_st
 Struct nvmlSample_st
 Struct nvmlUtilization_st
 Struct nvmlViolationTime_st

2.2 Enums

Enum nvmlBrandType_enum
 Enum nvmlClockId_enum
 Enum nvmlClockType_enum
 Enum nvmlEnableState_enum
 Enum nvmlInforomObject_enum
 Enum nvmlPerfPolicyType_enum
 Enum nvmlPStates_enum
 Enum nvmlReturn_enum
 Enum nvmlSamplingType_enum
 Enum nvmlTemperatureSensors_enum
 Enum nvmlTemperatureThresholds_enum
 Enum nvmlValueType_enum

2.3 Unions

Union nvmlValue_st

2.4 Functions

Function nvmlDeviceGetAutoBoostedClocksEnabled
 Function nvmlDeviceGetBoardId
 Function nvmlDeviceGetBoardPartNumber
 Function nvmlDeviceGetBrand
 Function nvmlDeviceGetArchitecture
 Function nvmlDeviceGetClock
 Function nvmlDeviceGetClockInfo
 Function nvmlDeviceGetComputeRunningProcesses
 Function nvmlDeviceGetCount
 Function nvmlDeviceGetCudaComputeCapability
 Function nvmlDeviceGetCurrentClocksThrottleReasons
 Function nvmlDeviceGetCurrPcieLinkGeneration
 Function nvmlDeviceGetCurrPcieLinkWidth
 Function nvmlDeviceGetEncoderCapacity
 Function nvmlDeviceGetEnforcedPowerLimit
 Function nvmlDeviceGetFanSpee
 Function nvmlDeviceGetHandleByIndex
 Function nvmlDeviceGetHandleByPciBusId
 Function nvmlDeviceGetHandleBySerial
 Function nvmlDeviceGetHandleByUUID
 Function nvmlDeviceGetIndex

Function nvmlDeviceGetInforomConfigurationChecksum
 Function nvmlDeviceGetInforomImageVersion
 Function nvmlDeviceGetInforomVersion
 Function nvmlDeviceGetMaxClockInfo
 Function nvmlDeviceGetMaxCustomerBoostClock
 Function nvmlDeviceGetMaxPcieLinkGeneration
 Function nvmlDeviceGetMaxPcieLinkWidth
 Function nvmlDeviceGetMemoryInfo
 Function nvmlDeviceGetMinorNumber
 Function nvmlDeviceGetMultiGpuBoard
 Function nvmlDeviceGetName
 Function nvmlDeviceGetPciInfo
 Function nvmlDeviceGetPerformanceState
 Function nvmlDeviceGetPowerManagementDefaultLimit
 Function nvmlDeviceGetPowerManagementLimit
 Function nvmlDeviceGetPowerManagementLimitConstraints
 Function nvmlDeviceGetPowerManagementMode
 Function nvmlDeviceGetPowerState
 Function nvmlDeviceGetPowerUsage
 Function nvmlDeviceGetSamples
 Function nvmlDeviceGetSerial
 Function nvmlDeviceGetSupportedClocksThrottleReasons
 Function nvmlDeviceGetSupportedEventTypes
 Function nvmlDeviceGetSupportedMemoryClocks
 Function nvmlDeviceGetTemperature
 Function nvmlDeviceGetTemperatureThreshold
 Function nvmlDeviceGetTotalEnergyConsumption
 Function nvmlDeviceGetUtilizationRates
 Function nvmlDeviceGetUUID
 Function nvmlDeviceGetViolationStatus
 Function nvmlDeviceOnSameBoard
 Function nvmlDeviceRegisterEvents
 Function nvmlDeviceResetGpuLockedClocks
 Function nvmlDeviceSetAutoBoostedClocksEnabled
 Function nvmlDeviceSetDefaultAutoBoostedClocksEnabled
 Function nvmlDeviceSetGpuLockedClocks
 Function nvmlDeviceSetPowerManagementLimit
 Function nvmlDeviceValidateInforom
 Function nvmlErrorString
 Function nvmlEventSetCreate
 Function nvmlEventSetFree
 Function nvmlEventSetWait
 Function nvmlInit
 Function nvmlInitWithFlags
 Function nvmlShutdown
 Function nvmlSystemGetCudaDriverVersion
 Function nvmlSystemGetDriverVersion
 Function nvmlSystemGetNVMLVersion
 Function nvmlSystemGetProcessName

2.5 Defines

Define DECLDIR
 Define NVML_API_VERSION
 Define NVML_API_VERSION_STR
 Define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE

```

Define NVML_DEVICE_NAME_BUFFER_SIZE
Define NVML_DEVICE_PART_NUMBER_BUFFER_SIZE
Define NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE
Define NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE
Define NVML_DEVICE_PCI_BUS_ID_FMT
Define NVML_DEVICE_PCI_BUS_ID_FMT_ARGS
Define NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT
Define NVML_DEVICE_SERIAL_BUFFER_SIZE
Define NVML_DEVICE_UUID_BUFFER_SIZE
Define NVML_INIT_FLAG_NO_ATTACH
Define NVML_INIT_FLAG_NO_GPUS
Define NVML_MAX_CLUSTER_NUMBER
Define NVML_MAX_VFUNC_NUMBER
Define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE
Define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE
Define NVML_VALUE_NOT_AVAILABLE
Define nvmlClocksThrottleReasonAll
Define nvmlClocksThrottleReasonApplicationsClocksSetting
Define nvmlClocksThrottleReasonDisplayClockSetting
Define nvmlClocksThrottleReasonGpuIdle
Define nvmlClocksThrottleReasonHwPowerBrakeSlowdown
Define nvmlClocksThrottleReasonHwSlowdown
Define nvmlClocksThrottleReasonHwThermalSlowdown
Define nvmlClocksThrottleReasonNone
Define nvmlClocksThrottleReasonSwPowerCap
Define nvmlClocksThrottleReasonSwThermalSlowdown
Define nvmlClocksThrottleReasonSyncBoost
Define nvmlClocksThrottleReasonUserDefinedClocks
Define nvmlEventTypeAll
Define nvmlEventTypeClock
Define nvmlEventTypeDoubleBitEccError
Define nvmlEventTypeNone
Define nvmlEventTypePState
Define nvmlEventTypeSingleBitEccError
Define nvmlEventTypeXidCriticalError
Define nvmlFlagDefault
Define nvmlFlagForce

```

2.6 Typedefs

```

Typedef nvmlBrandType_t
Typedef nvmlClockId_t
Typedef nvmlClockType_t
Typedef nvmlClusterRemap_t
Typedef nvmlDevice_t
Typedef nvmlEnableState_t
Typedef nvmlEventData_t
Typedef nvmlEventSet_t
Typedef nvmlInforomObject_t
Typedef nvmlMemory_t
Typedef nvmlPciInfo_t
Typedef nvmlPerfPolicyType_t
Typedef nvmlProcessInfo_t
Typedef nvmlProcessUtilizationSample_t
Typedef nvmlPstates_t
Typedef nvmlReturn_t

```

```
Typedef nvmlSample_t  
Typedef nvmlSamplingType_t  
Typedef nvmlTemperatureSensors_t  
Typedef nvmlTemperatureThresholds_t  
Typedef nvmlUtilization_t  
Typedef nvmlValue_t  
Typedef nvmlValueType_t  
Typedef nvmlViolationTime_t
```

登临科技保密材料

LIBRARY API

1 Class Hierarchy

- [Struct nvmlClusterRemap_st](#)
- [Struct nvmlEventData_st](#)
- [Struct nvmlMemory_st](#)
- [Struct nvmlPciInfo_st](#)
- [Struct nvmlProcessInfo_st](#)
- [Struct nvmlProcessUtilizationSample_st](#)
- [Struct nvmlSample_st](#)
- [Struct nvmlUtilization_st](#)
- [Struct nvmlViolationTime_st](#)
- [Enum nvmlBrandType_enum](#)
- [Enum nvmlClockId_enum](#)
- [Enum nvmlClockType_enum](#)
- [Enum nvmlEnableState_enum](#)
- [Enum nvmlInforomObject_enum](#)
- [Enum nvmlPerfPolicyType_enum](#)
- [Enum nvmlPStates_enum](#)
- [Enum nvmlReturn_enum](#)
- [Enum nvmlSamplingType_enum](#)
- [Enum nvmlTemperatureSensors_enum](#)
- [Enum nvmlTemperatureThresholds_enum](#)
- [Enum nvmlValueType_enum](#)
- [Union nvmlValue_st](#)

2 Full API

2.1 Classes and Structs

Struct `nvmlClusterRemap_st`

- Defined in file `dlml.h`

Struct Documentation

```
struct nvmlClusterRemap_st
```

Struct to hold cluster remap states

Public Members

- unsigned int `clusterId[NVML_MAX_CLUSTER_NUMBER]`
Physical cluster id(s) in this vFunc(guest)
- unsigned int `clusterCount`
Number of physical clusters in this vFunc
- struct `nvmlClusterRemap_st::[anonymous] vFunc[NVML_MAX_VFUNC_NUMBER]`
vFunc definition
- unsigned int `vFuncCount`
Total vFunc number

Struct `nvmlEventData_st`

- Defined in file `dlml.h`

Struct Documentation

```
struct nvmlEventData_st
```

Information about occurred event

Public Members

- `nvmlDevice_t device`
Specific device where the event occurred
- unsigned long long `eventType`
Information about what specific event occurred
- unsigned long long `eventData`
Stores last XID error for the device in the event of `nvmlEventTypeXidCriticalError`

Struct nvmlMemory_st

- Defined in file dlml.h

Struct Documentation

```
struct nvmlMemory_st
```

Memory allocation information for a device.

Public Members

- unsigned long long `total`
Total installed memory (in bytes)
- unsigned long long `free`
Unallocated memory (in bytes)
- unsigned long long `used`
Allocated memory (in bytes). Note that the driver/GPU always sets aside a small amount of memory for bookkeeping
- unsigned long long `clusterUsed[NVML_MAX_CLUSTER_NUMBER]`
Per cluster allocated memory (in bytes)

Struct nvmlPciInfo_st

- Defined in file dlml.h

Struct Documentation

```
struct nvmlPciInfo_st
```

PCI information about a GPU device

Public Members

- 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)

Struct `nvmlProcessInfo_st`

- Defined in file `dlml.h`

Struct Documentation

```
struct nvmlProcessInfo_st
```

Information about running compute processes on the GPU

Public Members

- unsigned int `pid`
Process ID
- unsigned int `task_id`
GPU task ID
- unsigned long long `usedGpuMemory`
Amount of used GPU memory in bytes
- unsigned long long `clusterUsedGpuMemory[NVML_MAX_CLUSTER_NUMBER]`
Per cluster used GPU memory in bytes

Struct `nvmlProcessUtilizationSample_st`

- Defined in file `dlml.h`

Struct Documentation

```
struct nvmlProcessUtilizationSample_st
```

Structure to store utilization value and process Id

Public Members

- unsigned int `pid`
PID of process
- unsigned long long `timeStamp`
CPU Timestamp in microseconds
- unsigned int `smUtil`
SM (3D/Compute) Util Value

- unsigned int `memUtil`
Frame Buffer Memory Util Value
- unsigned int `encUtil`
Encoder Util Value
- unsigned int `decUtil`
Decoder Util Value

Struct `nvmlSample_st`

- Defined in file `dlml.h`

Struct Documentation

```
struct nvmlSample_st
```

Information for Sample

Public Members

- unsigned long long `timestamp`
CPU Timestamp in microseconds
- `nvmlValue_t` `sampleValue`
Sample Value

Struct `nvmlUtilization_st`

- Defined in file `dlml.h`

Struct Documentation

```
struct nvmlUtilization_st
```

Utilization information for a device. Each sample period may be between 1 second and 1/6 second, depending on the product being queried

Public Members

- unsigned int `gpu`
Percent of time over the past sample period during which one or more kernels was executing on the GPU
- unsigned int `memory`
Percent of time over the past sample period during which global (device) memory was being read or written

Struct nvmlViolationTime_st

- Defined in file dlml.h

Struct Documentation

```
struct nvmlViolationTime_st
```

Struct to hold perf policy violation status data

Public Members

- unsigned long long `referenceTime`
referenceTime represents CPU timestamp in microseconds
- unsigned long long `violationTime`
violationTime in Nanoseconds

2.2 Enums

Enum nvmlBrandType_enum

- Defined in file dlml.h

Enum Documentation

```
enum nvmlBrandType_enum
```

The Brand of the GPU

Values:

```
enumerator NVML_BRAND_UNKNOWN = 0
```

```
enumerator NVML_BRAND_GOLDWASSER = 1
```

```
enumerator NVML_BRAND_COUNT
```

Enum nvmlClockId_enum

- Defined in file dlml.h

Enum Documentation

```
enum nvmlClockId_enum
```

Clock Ids. These are used in combination with [nvmlClockType_t](#) to specify a single clock value

Values:

- `enumerator NVML_CLOCK_ID_CURRENT = 0`

Current actual clock value

- `enumerator NVML_CLOCK_ID_APP_CLOCK_TARGET = 1`

Target application clock

- `enumerator NVML_CLOCK_ID_APP_CLOCK_DEFAULT = 2`

Default application clock target

- `enumerator NVML_CLOCK_ID_CUSTOMER_BOOST_MAX = 3`

OEM-defined maximum clock rate

- `enumerator NVML_CLOCK_ID_COUNT`

Count of Clock Ids

Enum nvmlClockType_enum

- Defined in file dlml.h

Enum Documentation

```
enum nvmlClockType_enum
```

Clock types

All speeds are in Mhz.

Values:

- `enumerator NVML_CLOCK_CORE = 0`
Core(Compute) clock domain
- `enumerator NVML_CLOCK_TE = 1`
Tensor Engine clock domain
- `enumerator NVML_CLOCK_MEM = 2`
Memory clock domain
- `enumerator NVML_CLOCK_VIDEO = 3`
Video encoder/decoder clock domain
- `enumerator NVML_CLOCK_COUNT`
Count of clock types

Enum nvmlEnableState_enum

- Defined in file dlml.h

Enum Documentation

`enum nvmlEnableState_enum`

Generic enable/disable enum

Values:

- `enumerator NVML_FEATURE_DISABLED = 0`
Feature disabled
- `enumerator NVML_FEATURE_ENABLED = 1`
Feature enabled

Enum nvmlInforomObject_enum

- Defined in file dlml.h

Enum Documentation

`enum nvmlInforomObject_enum`

Available infoROM objects

Values:

- `enumerator NVML_INFOROM_OEM = 0`
An object defined by OEM

- `enumerator NVML_INFOROM_ECC = 1`

The ECC object determining the level of ECC support

- `enumerator NVML_INFOROM_POWER = 2`

The power management object

- `enumerator NVML_INFOROM_COUNT`

This counts the number of infoROM objects the driver knows about

Enum nvmlPerfPolicyType_enum

- Defined in file dlml.h

Enum Documentation

`enum nvmlPerfPolicyType_enum`

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

Values:

- `enumerator NVML_PERF_POLICY_POWER = 0`
How long did power violations cause the GPU to be below application clocks
- `enumerator NVML_PERF_POLICY_THERMAL = 1`
How long did thermal violations cause the GPU to be below application clocks
- `enumerator NVML_PERF_POLICY_SYNC_BOOST = 2`
How long did sync boost cause the GPU to be below application clocks
- `enumerator NVML_PERF_POLICY_BOARD_LIMIT = 3`
How long did the board limit cause the GPU to be below application clocks
- `enumerator NVML_PERF_POLICY_LOW_UTILIZATION = 4`
How long did low utilization cause the GPU to be below application clocks
- `enumerator NVML_PERF_POLICY_RELIABILITY = 5`
How long did the board reliability limit cause the GPU to be below application clocks
- `enumerator NVML_PERF_POLICY_TOTAL_APP_CLOCKS = 10`
Total time the GPU was held below application clocks by any limiter (0 - 5 above)
- `enumerator NVML_PERF_POLICY_TOTAL_BASE_CLOCKS = 11`
Total time the GPU was held below base clocks
- `enumerator NVML_PERF_POLICY_COUNT`

Enum nvmlPStates_enum

- Defined in file dlml.h

Enum Documentation

`enum nvmlPStates_enum`

Allowed PStates

Values:

- `enumerator NVML_PSTATE_0` = 0
Performance state 0 Maximum Performance
- `enumerator NVML_PSTATE_1` = 1
Performance state 1
- `enumerator NVML_PSTATE_2` = 2
Performance state 2
- `enumerator NVML_PSTATE_3` = 3
Performance state 3
- `enumerator NVML_PSTATE_4` = 4
Performance state 4
- `enumerator NVML_PSTATE_5` = 5
Performance state 5
- `enumerator NVML_PSTATE_6` = 6
Performance state 6
- `enumerator NVML_PSTATE_7` = 7
Performance state 7
- `enumerator NVML_PSTATE_8` = 8
Performance state 8
- `enumerator NVML_PSTATE_9` = 9
Performance state 9
- `enumerator NVML_PSTATE_10` = 10
Performance state 10
- `enumerator NVML_PSTATE_11` = 11
Performance state 11
- `enumerator NVML_PSTATE_12` = 12
Performance state 12
- `enumerator NVML_PSTATE_13` = 13
Performance state 13
- `enumerator NVML_PSTATE_14` = 14

Performance state 14

- `enumerator NVML_PSTATE_15` = 15

Performance state 15 Minimum Performance

- `enumerator NVML_PSTATE_UNKNOWN` = 32

Unknown performance state

Enum nvmlReturn_enum

- Defined in file dlml.h

Enum Documentation

`enum nvmlReturn_enum`

Return values for NVML API calls

Values:

- `enumerator NVML_SUCCESS` = 0
The operation was successful.
- `enumerator NVML_ERROR_UNINITIALIZED` = 1
NVML was not first initialized with `nvmlInit()`.
- `enumerator NVML_ERROR_INVALID_ARGUMENT` = 2
A supplied argument is invalid.
- `enumerator NVML_ERROR_NOT_SUPPORTED` = 3
The requested operation is not available on target device.
- `enumerator NVML_ERROR_NO_PERMISSION` = 4
The current user does not have permission for operation.
- `enumerator NVML_ERROR_ALREADY_INITIALIZED` = 5
Deprecated: Multiple initializations are now allowed through ref counting.
- `enumerator NVML_ERROR_NOT_FOUND` = 6
A query to find an object was unsuccessful.
- `enumerator NVML_ERROR_INSUFFICIENT_SIZE` = 7
An input argument is not large enough.
- `enumerator NVML_ERROR_INSUFFICIENT_POWER` = 8
A device's external power cables are not properly attached.
- `enumerator NVML_ERROR_DRIVER_NOT_LOADED` = 9
denglin driver is not loaded.
- `enumerator NVML_ERROR_TIMEOUT` = 10
User provided timeout passed.
- `enumerator NVML_ERROR_IRQ_ISSUE` = 11

denglin Kernel detected an interrupt issue with a GPU.

- `enumerator NVML_ERROR_LIBRARY_NOT_FOUND` = 12

NVML Shared Library couldn't be found or loaded.

- `enumerator NVML_ERROR_FUNCTION_NOT_FOUND` = 13

Local version of NVML doesn't implement this function.

- `enumerator NVML_ERROR_CORRUPTED_INFOROM` = 14

infoROM is corrupted.

- `enumerator NVML_ERROR_GPU_IS_LOST` = 15

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

- `enumerator NVML_ERROR_RESET_REQUIRED` = 16

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

- `enumerator NVML_ERROR_OPERATING_SYSTEM` = 17

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

- `enumerator NVML_ERROR_LIB_RM_VERSION_MISMATCH` = 18

RM detects a driver/library version mismatch.

- `enumerator NVML_ERROR_IN_USE` = 19

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

- `enumerator NVML_ERROR_MEMORY` = 20

Insufficient memory.

- `enumerator NVML_ERROR_NO_DATA` = 21

No data.

- `enumerator NVML_ERROR_VGPU_ECC_NOT_SUPPORTED` = 22

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

- `enumerator NVML_ERROR_UNKNOWN` = 999

An internal driver error occurred.

Enum nvmlSamplingType_enum

- Defined in file dlml.h

Enum Documentation

`enum nvmlSamplingType_enum`

Represents Type of Sampling Event.

Values:

- `enumerator NVML_TOTAL_POWER_SAMPLES` = 0

To represent total power drawn by GPU

- `enumerator NVML_GPU_UTILIZATION_SAMPLES` = 1

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

- `enumerator NVML_MEMORY_UTILIZATION_SAMPLES = 2`

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

- `enumerator NVML_ENC_UTILIZATION_SAMPLES = 3`

To represent percent of time during which NVENC remains busy

- `enumerator NVML_DEC_UTILIZATION_SAMPLES = 4`

To represent percent of time during which NVDEC remains busy

- `enumerator NVML_PROCESSOR_CLK_SAMPLES = 5`

To represent processor clock samples

- `enumerator NVML_MEMORY_CLK_SAMPLES = 6`

To represent memory clock samples

- `enumerator NVML_SAMPLINGTYPE_COUNT`

Enum nvmlTemperatureSensors_enum

- Defined in file dlml.h

Enum Documentation

```
enum nvmlTemperatureSensors_enum
```

Temperature sensors

Values:

- `enumerator NVML_TEMPERATURE_GPU = 0`

Temperature sensor for the GPU die

- `enumerator NVML_TEMPERATURE_MEM = 1`

- `enumerator NVML_TEMPERATURE_COUNT`

Temperature sensor for the Memory

Enum nvmlTemperatureThresholds_enum

- Defined in file dlml.h

Enum Documentation

```
enum nvmlTemperatureThresholds_enum
```

Temperature thresholds

Values:

- `enumerator NVML_TEMPERATURE_THRESHOLD_SHUTDOWN = 0`

- `enumerator NVML_TEMPERATURE_THRESHOLD_SLOWDOWN = 1`

- `enumerator NVML_TEMPERATURE_THRESHOLD_MEM_MAX = 2`

- `enumerator NVML_TEMPERATURE_THRESHOLD_GPU_MAX = 3`
- `enumerator NVML_TEMPERATURE_THRESHOLD_COUNT`

Enum `nvmlValueType_enum`

- Defined in file `dlml.h`

Enum Documentation

`enum nvmlValueType_enum`

Represents the type for sample value returned

Values:

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

2.3 Unions

Union nvmlValue_st

- Defined in file dlml.h

Union Documentation

```
union nvmlValue_st
```

#include <dlml_doc.h> Union to represent different types of Value

Public Members

- 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.

2.4 Functions

Function `nvmlDeviceGetAutoBoostedClocksEnabled`

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the current state of Auto Boosted clocks on a device and stores it in *isEnabled*.

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.

Return

- `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

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

Function `nvmlDeviceGetBoardId`

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the device *boardId* from 0 to 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.

Return

- `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

Parameters

- `device`: The identifier of the target device
- `boardId`: Reference in which to return the device's board ID

Function nvmlDeviceGetBoardPartNumber

- Defined in file `dlml.h`

Function Documentation

```
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.

Return

- `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

Parameter

- `device`: Identifier of the target device
- `partNumber`: Reference to the buffer to return
- `length`: Length of the buffer reference

Function nvmlDeviceGetBrand

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the brand of this device.

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

Return

- `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

Parameters

- `device`: The identifier of the target device

- `type`: Reference in which to return the product brand type

Function `nvmlDeviceGetArchitecture`

- Defined in file `dlml.h`

Function Documentation

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

Get architecture for the device.

Return

- `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

Parameters

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

Function `nvmlDeviceGetClock`

- Defined in file `dlml.h`

Function Documentation

```
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.

Return

- `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

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

Function nvmlDeviceGetClockInfo

- Defined in file dlml.h

Function Documentation

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

Retrieves the current clock speeds for the device.

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

Return

- `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

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

Function nvmlDeviceGetComputeRunningProcesses

- Defined in file dlml.h

Function Documentation

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

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

This function returns information only about compute running processes (for example, CUDA application which have active context). Any graphics applications (for example, 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.

Return

- `NVML_SUCCESS` if `infoCount` and `infos` have been populated
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized

- `NVML_ERROR_INSUFFICIENT_SIZE` if `infoCount` indicates that the `infos` array is too small `infoCount` will contain minimal amount of space necessary for the call to complete
- `NVML_ERROR_INVALID_ARGUMENT` if device is invalid, either of `infoCount` or `infos` is NULL
- `NVML_ERROR_GPU_IS_LOST` if the target GPU has fallen off the bus or is otherwise inaccessible
- `NVML_ERROR_UNKNOWN` on any unexpected error

See [nvmlSystemGetProcessName](#).

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

Function nvmlDeviceGetCount

- Defined in file `dlml`.

Function Documentation

```
nvmlReturn_t DECLDIR nvmlDeviceGetCount (unsigned int *deviceCount)
```

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

Return

- `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

Parameters

- `deviceCount`: Reference in which to return the number of accessible devices

Function nvmlDeviceGetCudaComputeCapability

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the CUDA compute capability of the device.

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

Return

- `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

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

Function `nvmlDeviceGetCurrentClocksThrottleReasons`

- Defined in file `dlml.h`

Function Documentation

```
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.

Return

- `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 `NvmlClocksThrottleReasons`.

See [nvmlDeviceGetSupportedClocksThrottleReasons](#).

Parameters

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

Function `nvmlDeviceGetCurrPcieLinkGeneration`

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the current PCIe link generation.

Return

- `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

Parameters

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

Function nvmlDeviceGetCurrPcieLinkWidth

- Defined in file dlml.h

Function Documentation

```
nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkwidth (nvmlDevice_t device, unsigned int *currLinkwidth)
```

Retrieves the current PCIe link width.

Return

- `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

Parameters

- `device`: The identifier of the target device
- `currLinkwidth`: Reference in which to return the current PCIe link generation

Function nvmlDeviceGetEncoderCapacity

- Defined in file dlml.h

Function Documentation

```
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.

Return

- `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 encoderQuery-Type are invalid
- `NVML_ERROR_NOT_SUPPORTED` if device does not support the encoder specified in encode-QueryType
- `NVML_ERROR_GPU_IS_LOST` if the target GPU has fallen off the bus or is otherwise inaccessible
- `NVML_ERROR_UNKNOWN` on any unexpected error

Parameters

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

Function nvmlDeviceGetEnforcedPowerLimit

- Defined in file dlml.h

Function Documentation

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

Gets 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.

Return

- `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

Parameters

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

Function nvmlDeviceGetFanSpeed

- Defined in file dlml.h

Function Documentation

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

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

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

For all discrete products with dedicated fans.

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

Return

- `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

Parameters

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

Function `nvmlDeviceGetHandleByIndex`

- Defined in file `dlml.h`

Function Documentation

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

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

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

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

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

Return

- `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 denglin 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 [nvmlDeviceGetIndex](#).

See [nvmlDeviceGetCount](#).

Parameters

- `index`: The index of the target GPU, ≥ 0 and $< \text{accessibleDevices}$

- `device`: Reference in which to return the device handle

Function `nvmlDeviceGetHandleByPciBusId`

- Defined in file `dlml.h`

Function Documentation

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

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

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

Return

- `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 denglin 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

Parameters

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

Function `nvmlDeviceGetHandleBySerial`

- Defined in file `dlml.h`

Function Documentation

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

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

This number corresponds to the value printed directly on the board, and to the value returned by [nvmlDeviceGetSerial\(\)](#).

NVML may initialize additional GPUs as it searches for the target GPU.

Return

- `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 denglin 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 [nvmlDeviceGetSerial](#).

See [nvmlDeviceGetHandleByUUID](#).

Parameters

- `serial`: The board serial number of the target GPU
- `device`: Reference in which to return the device handle

Function nvmlDeviceGetHandleByUUID

- Defined in file `dlml.h`

Function Documentation

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

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

NVML may initialize additional GPUs as it searches for the target GPU.

Return

- `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 denglin 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 [nvmlDeviceGetUUID](#).

Parameters

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

Function nvmlDeviceGetIndex

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the NVML index of this device.

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

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

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

Return

- `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 [nvmlDeviceGetHandleByIndex\(\)](#).

See [nvmlDeviceGetCount\(\)](#).

Parameters

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

Function `nvmlDeviceGetInforomConfigurationChecksum`

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the checksum of the configuration stored in the device's 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 (for example, disable/enable ECC).

Return

- `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

Parameters

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

Function nvmlDeviceGetInforomImageVersion

- Defined in file dlml.h

Function Documentation

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

Retrieves the global infoROM image version.

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`.

Return

- `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 [nvmlDeviceGetInforomVersion](#).

Parameters

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

Function nvmlDeviceGetInforomVersion

- Defined in file dlml.h

Function Documentation

```
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.

See `nvmlConstants::NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE`.

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

Return

- `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 [nvmlDeviceGetInfoROMImageVersion](#).

Parameters

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

Function nvmlDeviceGetMaxClockInfo

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the maximum clock speeds for the device.

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

Note: GPUs P0 clocks (reported by [nvmlDeviceGetClockInfo](#)) can differ from max clocks by few MHz.

Return

- `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

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

Function nvmlDeviceGetMaxCustomerBoostClock

- Defined in file `dlml.h`

Function Documentation

```
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.

Return

- `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

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

Function `nvmlDeviceGetMaxPcieLinkGeneration`

- Defined in file `dlml.h`

Function Documentation

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

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

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

Return

- `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

Parameters

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

Function `nvmlDeviceGetMaxPcieLinkWidth`

- Defined in file `dlml.h`

Function Documentation

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

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

That is, 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.

Return

- `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

Parameters

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

Function nvmlDeviceGetMemoryInfo

- Defined in file dlml.h

Function Documentation

```
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.

Enabling ECC reduces the amount of total available memory, due to the extra required parity bits.

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.

Return

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

Parameters

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

Function nvmlDeviceGetMinorNumber

- Defined in file dlml.h

Function Documentation

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

Retrieves minor number for the device.

Return

- `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

Parameters

- `device`: The identifier of the target device
- `minorNumber`: Reference in which to return the minor number for the device

Function nvmlDeviceGetMultiGpuBoard

- Defined in file dlml.h

Function Documentation

```
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.

Return

- `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

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

Function nvmlDeviceGetName

- Defined in file dlml.h

Function Documentation

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

Retrieves the name of this device.

The name is an alphanumeric string that denotes a particular product. It will not exceed 64 characters in length (including the NULL terminator). See `nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE`.

Return

- `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

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

Function nvmlDeviceGetPciInfo

- Defined in file dlml.h

Function Documentation

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

Retrieves the PCI attributes of this device.

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

Return

- `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

Parameters

- `device`: The identifier of the target device
- `pci`: Reference in which to return the PCI information

Function nvmlDeviceGetPerformanceState

- Defined in file dlml.h

Function Documentation

```
nvmlReturn_t DECLDIR nvmlDeviceGetPerformanceState (nvmlDevice_t device, nvmlPstates_t *pstate)
```

Retrieves the current performance state for the device.

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

Return

- `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

Parameters

- `device`: The identifier of the target device
- `pstate`: Reference in which to return the performance state reading

Function nvmlDeviceGetPowerManagementDefaultLimit

- Defined in file dlml.h

Function Documentation

```
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.

Return

- `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

Parameters

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

Function nvmlDeviceGetPowerManagementLimit

- Defined in file dlml.h

Function Documentation

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

Retrieves the power management limit associated with this device.

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](#).

Return

- `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

Parameters

- `device`: The identifier of the target device
- `limit`: Reference in which to return the power management limit in milliwatts

Function nvmlDeviceGetPowerManagementLimitConstraints

- Defined in file dlml.h

Function Documentation

```
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.

Return

- `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 [nvmlDeviceSetPowerManagementLimit](#).

Parameters

- `device`: The identifier of the target device

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

Function `nvmlDeviceGetPowerManagementMode`

- Defined in file `dlml.h`

Function Documentation

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

This API has been deprecated.

Retrieves the power management mode associated with this device

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.

Return

- `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

Parameters

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

Function `nvmlDeviceGetPowerState`

- Defined in file `dlml.h`

Function Documentation

```
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.

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

Return

- `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

Parameters

- `device`: The identifier of the target device
- `pstate`: Reference in which to return the performance state reading

Function `nvmlDeviceGetPowerUsage`

- Defined in file `dlml.h`

Function Documentation

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

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

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

Return

- `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

Parameters

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

Function `nvmlDeviceGetSamples`

- Defined in file `dlml.h`

Function Documentation

```
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.

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

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

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

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

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

Return

- `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

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 `nvml-SampleVal_t`
- `sampleCount`: Reference to provide the number of elements which can be queried in samples array
- `samples`: Reference in which samples are returned

Function nvmlDeviceGetSerial

- Defined in file `dlml.h`

Function Documentation

```
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.

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`.

Return

- `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

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

Function `nvmlDeviceGetSupportedClocksThrottleReasons`

- Defined in file `dlml.h`

Function Documentation

```
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).

Return

- `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 `NvmlClocksThrottleReasons`.

See [nvmlDeviceGetCurrentClocksThrottleReasons](#).

Parameters

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

Function `nvmlDeviceGetSupportedEventTypes`

- Defined in file `dlml.h`

Function Documentation

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

Returns information about events supported on device.

Return

- `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 Event Types.

See [nvmlDeviceRegisterEvents](#).

Parameters

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

Function nvmlDeviceGetSupportedMemoryClocks

- Defined in file dlml.h

Function Documentation

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

Retrieves the list of possible memory clocks.

Return

- `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 [nvmlDeviceSetApplicationsClocks](#).

See [nvmlDeviceGetSupportedGraphicsClocks](#).

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

Function nvmlDeviceGetTemperature

- Defined in file dlml.h

Function Documentation

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

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

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

Return

- `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

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

Function nvmlDeviceGetTemperatureThreshold

- Defined in file dlml.h

Function Documentation

```
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.

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

Return

- `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

Parameters

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

Function nvmlDeviceGetTotalEnergyConsumption

- Defined in file dlml.h

Function Documentation

```
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.

Return

- `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

Parameters

- `device`: The identifier of the target device
- `energy`: Reference in which to return the energy consumption information

Function nvmlDeviceGetUtilizationRates

- Defined in file dlml.h

Function Documentation

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

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

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.

Return

- `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

Parameters

- `device`: The identifier of the target device

- `utilization`: Reference in which to return the utilization information

Function nvmlDeviceGetUUID

- Defined in file dlml.h

Function Documentation

```
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.

The UUID is a globally unique identifier. It does NOT correspond to any identifier printed on the board. It will not exceed 80 characters in length (including the NULL terminator). See `nvmlConstants::NVML_DEVICE_UUID_BUFFER_SIZE`.

Return

- `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

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

Function nvmlDeviceGetViolationStatus

- Defined in file dlml.h

Function Documentation

```
nvmlReturn_t DECLDIR nvmlDeviceGetViolationStatus (nvmlDevice_t device, nvmlPerfPolicyType, 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.

Return

- `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

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

Function `nvmlDeviceOnSameBoard`

- Defined in file `dlml.h`

Function Documentation

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

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

For all fully supported products

Return

- `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

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

Function `nvmlDeviceRegisterEvents`

- Defined in file `dlml.h`

Function Documentation

```
nvmlReturn_t DECLDIR nvmlDeviceRegisterEvents (nvmlDevice_t device, unsigned long long
*eventTypes)
```

Starts recording of events on a specified devices and add the events to specified [nvmlEventSet_t](#).

Power capping events are available only on Power Management enabled devices (see [nvmlDeviceGetPowerManagementMode](#)).

IMPORTANT: Operations on set are not thread safe.

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

If function reports NVML_ERROR_UNKNOWN, event set is in undefined state and should be freed. If function reports NVML_ERROR_NOT_SUPPORTED, event set can still be used. None of the requested eventTypes are registered in that case.

Return

- 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 Event Types.

See [nvmlDeviceGetSupportedEventTypes](#).

See [nvmlEventSetWait](#).

See [nvmlEventSetFree](#).

Parameters

- device : The identifier of the target device
- eventTypes : Bitmask of Event Types to record
- set : Set to which add new event types

Function nvmlDeviceResetGpuLockedClocks

- Defined in file dlml.h

Function Documentation

```
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.

See [nvmlDeviceSetGpuLockedClocks](#).

Return

- 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

Parameters

- `device`: The identifier of the target device

Function `nvmlDeviceSetAutoBoostedClocksEnabled`

- Defined in file `dlml.h`

Function Documentation

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

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

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.

Note: Persistence Mode is required to modify current Auto Boost settings, therefore, it must be enabled.

Return

- `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

Parameters

- `device`: The identifier of the target device
- `enabled`: What state to try to set Auto Boosted clocks of the target device to

Function `nvmlDeviceSetDefaultAutoBoostedClocksEnabled`

- Defined in file `dlml.h`

Function Documentation

```
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 (for example, CUDA application which have an active context) are running.

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.

Return

- `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

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.

Function nvmlDeviceSetGpuLockedClocks

- Defined in file dlml.h

Function Documentation

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

Sets 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 `nvmlDeviceSetApplicationsClocks`.

Can be used as a setting to request constant performance.

Requires root/admin permissions.

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

Return

- `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

Parameters

- `device`: The identifier of the target device

- `minGpuClockMHz`: Requested minimum gpu clock in MHz
- `maxGpuClockMHz`: Requested maximum gpu clock in MHz

Function `nvmlDeviceSetPowerManagementLimit`

- Defined in file `dlml.h`

Function Documentation

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

Sets new power limit of this device.

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.

Return

- `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 [nvmlDeviceGetPowerManagementLimitConstraints](#).

See [nvmlDeviceGetPowerManagementDefaultLimit](#).

Parameters

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

Function `nvmlDeviceValidateInforom`

- Defined in file `dlml.h`

Function Documentation

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

Reads the infoROM from the flash and verifies the checksums.

Return

- `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

Parameters

- `device`: The identifier of the target device

Function `nvmlErrorString`

- Defined in file `dlml.h`

Function Documentation

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

Helper method for converting NVML error codes into readable strings.

Return

String representation of the error

Parameters

- `result`: NVML error code to convert

Function `nvmlEventSetCreate`

- Defined in file `dlml.h`

Function Documentation

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

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

Return

- `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 [nvmlEventSetFree](#).

Parameters

- `set`: Reference in which to return the event handle

Function nvmlEventSetFree

- Defined in file dlml.h

Function Documentation

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

Releases events in the set.

Return

- `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 [nvmlDeviceRegisterEvents](#).

Parameters

- `set`: Reference to events to be released

Function nvmlEventSetWait

- Defined in file dlml.h

Function Documentation

```
nvmlReturn_t DECLDIR nvmlEventSetWait (nvmlEventSet_t set, nvmlEventData_t *data, unsigned int timeouts)
```

Waits on events and delivers events.

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 (for example, when interrupt arrives).

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

Return

- `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 Event Types.

See [nvmlDeviceRegisterEvents](#).

Parameters

- `set`: Reference to set of events to wait on

- `data`: Reference in which to return event data
- `timeouts`: Maximum amount of wait time in milliseconds for registered event

Function `nvmlInit`

- Defined in file `dlml.h`

Function Documentation

```
nvmlReturn_t DECLDIR nvmlInit (void)
```

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

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.

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.

Return

- `NVML_SUCCESS` if NVML has been properly initialized
- `NVML_ERROR_DRIVER_NOT_LOADED` if denglin 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

Function `nvmlInitWithFlags`

- Defined in file `dlml.h`

Function Documentation

```
nvmlReturn_t DECLDIR nvmlInitWithFlags (unsigned int flags)
```

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

Return

- `NVML_SUCCESS` if NVML has been properly initialized
- `NVML_ERROR_DRIVER_NOT_LOADED` if denglin 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

Function `nvmlShutdown`

- Defined in file `dlml.h`

Function Documentation

```
nvmlReturn_t DECLDIR nvmlShutdown (void)
```

Shuts down NVML by releasing all GPU resources previously allocated with `nvmlInit()`.

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

Return

- `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

Function `nvmlSystemGetCudaDriverVersion`

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the version of the CUDA driver.

The returned CUDA driver version is the same as the CUDA API `cuDriverGetVersion()` would return on the system.

Return

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

Parameters

- `cudaDriverVersion`: Reference in which to return the version identifier

Function `nvmlSystemGetDriverVersion`

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the version of the system's graphics driver.

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

Return

- `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

Parameters

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

Function `nvmlSystemGetNVMLVersion`

- Defined in file `dlml.h`

Function Documentation

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

Retrieves the version of the NVML library.

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

Return

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

Parameters

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

Function `nvmlSystemGetProcessName`

- Defined in file `dlml.h`

Function Documentation

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

Gets name of the process with provided process id.

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

Return

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

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

2.5 Defines

Define DECLDIR

- Defined in file dlml.h

Define Documentation

DECLDIR

Define NVML_API_VERSION

- Defined in file dlml.h

Define Documentation

NVML_API_VERSION

NVML API versioning support

Define NVML_API_VERSION_STR

- Defined in file dlml.h

Define Documentation

NVML_API_VERSION_STR

Define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE

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

Define NVML_DEVICE_NAME_BUFFER_SIZE

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_NAME_BUFFER_SIZE

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

Define NVML_DEVICE_PART_NUMBER_BUFFER_SIZE

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_PART_NUMBER_BUFFER_SIZE

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

Define NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE

Buffer size guaranteed to be large enough for pci bus id

Define NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_PCI_BUS_ID_BUFFER_V2_SIZE

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

Define NVML_DEVICE_PCI_BUS_ID_FMT

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_PCI_BUS_ID_FMT

PCI format string for nvmlPciInfo_t::busId

Define NVML_DEVICE_PCI_BUS_ID_FMT_ARGS

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_PCI_BUS_ID_FMT_ARGS(pciInfo)

Utility macro for filling the pci bus id format from a nvmlPciInfo_t

Define NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_PCI_BUS_ID_LEGACY_FMT

PCI format string for nvmlPciInfo_t::busIdLegacy

Define NVML_DEVICE_SERIAL_BUFFER_SIZE

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_SERIAL_BUFFER_SIZE

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

Define NVML_DEVICE_UUID_BUFFER_SIZE

- Defined in file dlml.h

Define Documentation

NVML_DEVICE_UUID_BUFFER_SIZE

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

Define NVML_INIT_FLAG_NO_ATTACH

- Defined in file dlml.h

Define Documentation

NVML_INIT_FLAG_NO_ATTACH

Don't attach GPUs.

Define NVML_INIT_FLAG_NO_GPUS

- Defined in file dlml.h

Define Documentation

NVML_INIT_FLAG_NO_GPUS

Don't fail nvmlInit() when no GPUs are found.

Define NVML_MAX_CLUSTER_NUMBER

- Defined in file dlml.h

Define Documentation

NVML_MAX_CLUSTER_NUMBER

Max cluster number

Define NVML_MAX_VFUNC_NUMBER

- Defined in file dlml.h

Define Documentation

NVML_MAX_VFUNC_NUMBER

Max vfunc (ie, vgpu or cluster group) number

Define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE

- Defined in file dlml.h

Define Documentation

NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE

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

Define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE

- Defined in file dlml.h

Define Documentation

NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE

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

Define NVML_VALUE_NOT_AVAILABLE

- Defined in file dlml.h

Define Documentation

NVML_VALUE_NOT_AVAILABLE

Special constant that some fields take when they are not available. Used when only part of the struct is not available.

Each structure explicitly states when to check for this value.

Define nvmlClocksThrottleReasonAll

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonAll`

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

Define nvmlClocksThrottleReasonApplicationsClocksSetting

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonApplicationsClocksSetting`

GPU clocks are limited by current setting of applications clocks.

See `nvmlDeviceSetApplicationsClocks`.

See `nvmlDeviceGetApplicationsClock`.

Define nvmlClocksThrottleReasonDisplayClockSetting

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonDisplayClockSetting`

GPU clocks are limited by current setting of Display clocks.

See bug 1997531.

Define nvmlClocksThrottleReasonGpuIdle

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonGpuIdle`

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

Note: This limiter may be removed in a later release.

Define nvmlClocksThrottleReasonHwPowerBrakeSlowdown

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonHwPowerBrakeSlowdown`

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

This is an indicator of:

- External Power Brake Assertion being triggered (for example, by the system power supply)

See [nvmlDeviceGetTemperature](#).

See [nvmlDeviceGetTemperatureThreshold](#).

See [nvmlDeviceGetPowerUsage](#).

Define nvmlClocksThrottleReasonHwSlowdown

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonHwSlowdown`

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

This is an indicator of:

- temperature being too high.
- External Power Brake Assertion is triggered (for example, 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 [nvmlDeviceGetTemperature](#).

See [nvmlDeviceGetTemperatureThreshold](#).

See [nvmlDeviceGetPowerUsage](#).

Define nvmlClocksThrottleReasonHwThermalSlowdown

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonHwThermalSlowdown`

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

This is an indicator of:

- temperature being too high

See [nvmlDeviceGetTemperature](#).

See [nvmlDeviceGetTemperatureThreshold](#).

See [nvmlDeviceGetPowerUsage](#).

Define nvmlClocksThrottleReasonNone

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonNone`

Bit mask representing no clocks throttling

Clocks are as high as possible.

Define nvmlClocksThrottleReasonSwPowerCap

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonSwPowerCap`

SW Power Scaling algorithm is reducing the clocks below requested clocks

See [nvmlDeviceGetPowerUsage](#).

See [nvmlDeviceSetPowerManagementLimit](#).

See [nvmlDeviceGetPowerManagementLimit](#).

Define nvmlClocksThrottleReasonSwThermalSlowdown

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonSwThermalSlowdown`

SW Thermal Slowdown

This is an indicator of one or more of the following:

- Current GPU temperature above the GPU Max Operating Temperature
- Current memory temperature above the Memory Max Operating Temperature

Define nvmlClocksThrottleReasonSyncBoost

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonSyncBoost`

Sync Boost

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

Define nvmlClocksThrottleReasonUserDefinedClocks

- Defined in file dlml.h

Define Documentation

`nvmlClocksThrottleReasonUserDefinedClocks`

Define nvmlEventTypeAll

- Defined in file dlml.h

Define Documentation

`nvmlEventTypeAll`

Mask of all events

Define nvmlEventTypeClock

- Defined in file dlml.h

Define Documentation

`nvmlEventTypeClock`

Event about clock changes

Define nvmlEventTypeDoubleBitEccError

- Defined in file dlml.h

Define Documentation

`nvmlEventTypeDoubleBitEccError`

Event about double bit ECC errors

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

Define `nvmlEventTypeNone`

- Defined in file `dlml.h`

Define Documentation

`nvmlEventTypeNone`

Mask with no events

Define `nvmlEventTypePState`

- Defined in file `dlml.h`

Define Documentation

`nvmlEventTypePState`

Event about PState changes

Note: 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, GPU should stay in P0 for the duration of the execution of the compute process.

Define `nvmlEventTypeSingleBitEccError`

- Defined in file `dlml.h`

Define Documentation

`nvmlEventTypeSingleBitEccError`

Event about single bit ECC errors

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

Define `nvmlEventTypeXidCriticalError`

- Defined in file `dlml.h`

Define Documentation

`nvmlEventTypeXidCriticalError`

Event that Xid critical error occurred

Define nvmlFlagDefault

- Defined in file dlml.h

Define Documentation

`nvmlFlagDefault`

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

Define nvmlFlagForce

- Defined in file dlml.h

Define Documentation

`nvmlFlagForce`

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

2.6 Typedefs

Typedef nvmlBrandType_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlBrandType_enum nvmlBrandType_t
```

The Brand of the GPU

Typedef nvmlClockId_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlClockId_enum nvmlClockId_t
```

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

Typedef nvmlClockType_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlClockType_enum nvmlClockType_t
```

Clock types

All speeds are in Mhz.

Typedef nvmlClusterRemap_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlClusterRemap_st nvmlClusterRemap_t
```

Struct to hold cluster remap states

Typedef nvmlDevice_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlDevice_st *nvmlDevice_t
```

Typedef nvmlEnableState_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlEnableState_enum nvmlEnableState_t
```

Generic enable/disable enum

Typedef nvmlEventData_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlEventData_st nvmlEventData_t
```

Information about occurred event

Typedef nvmlEventSet_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlEventSet_st *nvmlEventSet_t
```

Handle to an event set

Typedef nvmlInforomObject_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlInforomObject_enum nvmlInforomObject_t
```

Available infoROM objects

Typedef nvmlMemory_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlMemory_st nvmlMemory_t
```

Memory allocation information for a device

Typedef nvmlPciInfo_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlPciInfo_st nvmlPciInfo_t
```

PCI information about a GPU device

Typedef nvmlPerfPolicyType_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlPerfPolicyType_enum nvmlPerfPolicyType_t
```

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

Typedef nvmlProcessInfo_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlProcessInfo_st nvmlProcessInfo_t
```

Information about running compute processes on the GPU

Typedef nvmlProcessUtilizationSample_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlProcessUtilizationSample_st nvmlProcessUtilizationSample_t
```

Structure to store utilization value and process Id

Typedef nvmlPstates_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlPStates_enum nvmlPstates_t
```

Allowed PStates

Typedef nvmlReturn_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlReturn_enum nvmlReturn_t
```

Return values for NVML API calls

Typedef nvmlSample_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlSample_st nvmlSample_t
```

Information for Sample

Typedef nvmlSamplingType_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlSamplingType_enum nvmlSamplingType_t
```

Represents Type of Sampling Event

Typedef nvmlTemperatureSensors_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlTemperatureSensors_enum nvmlTemperatureSensors_t
```

Temperature sensors

Typedef nvmlTemperatureThresholds_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlTemperatureThresholds_enum nvmlTemperatureThresholds_t
```

Temperature thresholds

Typedef nvmlUtilization_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlUtilization_st nvmlUtilization_t
```

Utilization information for a device. Each sample period may be between 1 second and 1/6 second, depending on the product being queried.

Typedef nvmlValue_t

- Defined in file dlml.h

Typedef Documentation

```
typedef union nvmlValue_st nvmlValue_t
```

Union to represent different types of Value

Typedef nvmlValueType_t

- Defined in file dlml.h

Typedef Documentation

```
typedef enum nvmlValueType_enum nvmlValueType_t
```

Represents the type for sample value returned

Typedef nvmlViolationTime_t

- Defined in file dlml.h

Typedef Documentation

```
typedef struct nvmlViolationTime_st nvmlViolationTime_t
```

Struct to hold perf policy violation status data

- genindex
- search