Contents

1 Known issues in the current version of NVML library ........................................ 1

2 Change log of NVML library .............................................................................. 3
  2.1 Changes between NVML v4.304 and v5.319 RC ........................................... 4
  2.2 Changes between NVML v4.304 RC and v4.304 Production ......................... 4
  2.3 Changes between NVML v3.295 and v4.304 RC ........................................... 4
  2.4 Changes between NVML v2.285 and v3.295 .................................................. 5
  2.5 Changes between NVML v1.0 and v2.285 ..................................................... 5

3 Deprecated List ................................................................................................. 7

4 Module Index .................................................................................................... 9
  4.1 Modules ......................................................................................................... 9

5 Data Structure Index ........................................................................................ 11
  5.1 Data Structures ............................................................................................ 11

6 Module Documentation ..................................................................................... 13
  6.1 Device Structs ............................................................................................... 13
    6.1.1 Define Documentation ........................................................................... 13
      6.1.1.1 NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE ............................... 13
      6.1.1.2 NVML_VALUE_NOT_AVAILABLE .............................................. 13
  6.2 Device Enums ............................................................................................... 14
    6.2.1 Define Documentation ........................................................................... 16
      6.2.1.1 NVML_DOUBLE_BIT_ECC ......................................................... 16
      6.2.1.2 NVML_SINGLE_BIT_ECC ......................................................... 16
      6.2.1.3 nvmlEccBitType_t ....................................................................... 16
    6.2.2 Enumeration Type Documentation ....................................................... 16
      6.2.2.1 nvmlClockType_t ....................................................................... 16
      6.2.2.2 nvmlComputeMode_t ................................................................. 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.2.3</td>
<td>nvmlDriverModel_t</td>
</tr>
<tr>
<td>6.2.2.4</td>
<td>nvmlEccCounterType_t</td>
</tr>
<tr>
<td>6.2.2.5</td>
<td>nvmlEnableState_t</td>
</tr>
<tr>
<td>6.2.2.6</td>
<td>nvmlGpuOperationMode_t</td>
</tr>
<tr>
<td>6.2.2.7</td>
<td>nvmlInforomObject_t</td>
</tr>
<tr>
<td>6.2.2.8</td>
<td>nvmlMemoryErrorType_t</td>
</tr>
<tr>
<td>6.2.2.9</td>
<td>nvmlMemoryLocation_t</td>
</tr>
<tr>
<td>6.2.2.10</td>
<td>nvmlPageRetirementCause_t</td>
</tr>
<tr>
<td>6.2.2.11</td>
<td>nvmlPstates_t</td>
</tr>
<tr>
<td>6.2.2.12</td>
<td>nvmlReturn_t</td>
</tr>
<tr>
<td>6.2.2.13</td>
<td>nvmlTemperatureSensors_t</td>
</tr>
<tr>
<td>6.3</td>
<td>Unit Structs</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Enumeration Type Documentation</td>
</tr>
<tr>
<td>6.3.1.1</td>
<td>nvmlFanState_t</td>
</tr>
<tr>
<td>6.3.1.2</td>
<td>nvmlLedColor_t</td>
</tr>
<tr>
<td>6.4</td>
<td>Event Types</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Define Documentation</td>
</tr>
<tr>
<td>6.4.2.1</td>
<td>nvmlEventTypeClock</td>
</tr>
<tr>
<td>6.4.2.2</td>
<td>nvmlEventTypeDoubleBitEccError</td>
</tr>
<tr>
<td>6.4.2.3</td>
<td>nvmlEventTypePState</td>
</tr>
<tr>
<td>6.4.2.4</td>
<td>nvmlEventTypeSingleBitEccError</td>
</tr>
<tr>
<td>6.5</td>
<td>Accounting Statistics</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>6.5.2.1</td>
<td>nvmlDeviceClearAccountingPids</td>
</tr>
<tr>
<td>6.5.2.2</td>
<td>nvmlDeviceGetAccountingBufferSize</td>
</tr>
<tr>
<td>6.5.2.3</td>
<td>nvmlDeviceGetAccountingMode</td>
</tr>
<tr>
<td>6.5.2.4</td>
<td>nvmlDeviceGetAccountingPids</td>
</tr>
<tr>
<td>6.5.2.5</td>
<td>nvmlDeviceGetAccountingStats</td>
</tr>
<tr>
<td>6.5.2.6</td>
<td>nvmlDeviceSetAccountingMode</td>
</tr>
<tr>
<td>6.6</td>
<td>Initialization and Cleanup</td>
</tr>
<tr>
<td>6.6.1</td>
<td>Detailed Description</td>
</tr>
<tr>
<td>6.6.2</td>
<td>Function Documentation</td>
</tr>
<tr>
<td>6.6.2.1</td>
<td>nvmlInit</td>
</tr>
<tr>
<td>6.6.2.2</td>
<td>nvmlShutdown</td>
</tr>
<tr>
<td>6.7</td>
<td>Error reporting</td>
</tr>
</tbody>
</table>

NVIDIA Management Library
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7.1</td>
<td>Detailed Description</td>
<td>30</td>
</tr>
<tr>
<td>6.7.2</td>
<td>Function Documentation</td>
<td>30</td>
</tr>
<tr>
<td>6.7.2.1</td>
<td>nvmlErrorString</td>
<td>30</td>
</tr>
<tr>
<td>6.8</td>
<td>Constants</td>
<td>31</td>
</tr>
<tr>
<td>6.8.1</td>
<td>Define Documentation</td>
<td>31</td>
</tr>
<tr>
<td>6.8.1.1</td>
<td>NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE</td>
<td>31</td>
</tr>
<tr>
<td>6.8.1.2</td>
<td>NVML_DEVICE_NAME_BUFFER_SIZE</td>
<td>31</td>
</tr>
<tr>
<td>6.8.1.3</td>
<td>NVML DEVICE SERIAL BUFFER_SIZE</td>
<td>31</td>
</tr>
<tr>
<td>6.8.1.4</td>
<td>NVMLDEVICE_UUID_BUFFER_SIZE</td>
<td>31</td>
</tr>
<tr>
<td>6.8.1.5</td>
<td>NVML DEVICE VBIOS_VERSION_BUFFER_SIZE</td>
<td>31</td>
</tr>
<tr>
<td>6.8.1.6</td>
<td>NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE</td>
<td>31</td>
</tr>
<tr>
<td>6.8.1.7</td>
<td>NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE</td>
<td>31</td>
</tr>
<tr>
<td>6.9</td>
<td>System Queries</td>
<td>32</td>
</tr>
<tr>
<td>6.9.1</td>
<td>Detailed Description</td>
<td>32</td>
</tr>
<tr>
<td>6.9.2</td>
<td>Function Documentation</td>
<td>32</td>
</tr>
<tr>
<td>6.9.2.1</td>
<td>nvmlSystemGetDriverVersion</td>
<td>32</td>
</tr>
<tr>
<td>6.9.2.2</td>
<td>nvmlSystemGetNVMLVersion</td>
<td>32</td>
</tr>
<tr>
<td>6.9.2.3</td>
<td>nvmlSystemGetProcessName</td>
<td>33</td>
</tr>
<tr>
<td>6.10</td>
<td>Unit Queries</td>
<td>34</td>
</tr>
<tr>
<td>6.10.1</td>
<td>Detailed Description</td>
<td>34</td>
</tr>
<tr>
<td>6.10.2</td>
<td>Function Documentation</td>
<td>34</td>
</tr>
<tr>
<td>6.10.2.1</td>
<td>nvmlSystemGetHicVersion</td>
<td>34</td>
</tr>
<tr>
<td>6.10.2.2</td>
<td>nvmlUnitGetCount</td>
<td>34</td>
</tr>
<tr>
<td>6.10.2.3</td>
<td>nvmlUnitGetDevices</td>
<td>35</td>
</tr>
<tr>
<td>6.10.2.4</td>
<td>nvmlUnitGetFanSpeedInfo</td>
<td>35</td>
</tr>
<tr>
<td>6.10.2.5</td>
<td>nvmlUnitGetHandleByIndex</td>
<td>36</td>
</tr>
<tr>
<td>6.10.2.6</td>
<td>nvmlUnitGetLedState</td>
<td>36</td>
</tr>
<tr>
<td>6.10.2.7</td>
<td>nvmlUnitGetPsuInfo</td>
<td>36</td>
</tr>
<tr>
<td>6.10.2.8</td>
<td>nvmlUnitGetTemperature</td>
<td>37</td>
</tr>
<tr>
<td>6.10.2.9</td>
<td>nvmlUnitGetUnitInfo</td>
<td>37</td>
</tr>
<tr>
<td>6.11</td>
<td>Device Queries</td>
<td>38</td>
</tr>
<tr>
<td>6.11.1</td>
<td>Detailed Description</td>
<td>39</td>
</tr>
<tr>
<td>6.11.2</td>
<td>Function Documentation</td>
<td>39</td>
</tr>
<tr>
<td>6.11.2.1</td>
<td>nvmlDeviceGetApplicationsClock</td>
<td>39</td>
</tr>
<tr>
<td>6.11.2.2</td>
<td>nvmlDeviceGetClockInfo</td>
<td>40</td>
</tr>
<tr>
<td>6.11.2.3</td>
<td>nvmlDeviceGetComputeMode</td>
<td>40</td>
</tr>
<tr>
<td>6.11.2.4</td>
<td>nvmlDeviceGetComputeRunningProcesses</td>
<td>41</td>
</tr>
</tbody>
</table>

NVIDIA Management Library
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvmlDeviceGetCount</td>
<td>42</td>
</tr>
<tr>
<td>nvmlDeviceGetCurrentClocksThrottleReasons</td>
<td>42</td>
</tr>
<tr>
<td>nvmlDeviceGetCurrPcieLinkGeneration</td>
<td>43</td>
</tr>
<tr>
<td>nvmlDeviceGetCurrPcieLinkWidth</td>
<td>43</td>
</tr>
<tr>
<td>nvmlDeviceGetDefaultApplicationsClock</td>
<td>43</td>
</tr>
<tr>
<td>nvmlDeviceGetDetailedEccErrors</td>
<td>44</td>
</tr>
<tr>
<td>nvmlDeviceGetDisplayActive</td>
<td>45</td>
</tr>
<tr>
<td>nvmlDeviceGetDisplayMode</td>
<td>45</td>
</tr>
<tr>
<td>nvmlDeviceGetDriverModel</td>
<td>46</td>
</tr>
<tr>
<td>nvmlDeviceGetEccMode</td>
<td>46</td>
</tr>
<tr>
<td>nvmlDeviceGetFanSpeed</td>
<td>47</td>
</tr>
<tr>
<td>nvmlDeviceGetGpuOperationMode</td>
<td>47</td>
</tr>
<tr>
<td>nvmlDeviceGetHandleByIndex</td>
<td>48</td>
</tr>
<tr>
<td>nvmlDeviceGetHandleByPciBusId</td>
<td>49</td>
</tr>
<tr>
<td>nvmlDeviceGetHandleBySerial</td>
<td>50</td>
</tr>
<tr>
<td>nvmlDeviceGetHandleByUUID</td>
<td>50</td>
</tr>
<tr>
<td>nvmlDeviceGetIndex</td>
<td>51</td>
</tr>
<tr>
<td>nvmlDeviceGetInforomConfigurationChecksum</td>
<td>52</td>
</tr>
<tr>
<td>nvmlDeviceGetInforomImageVersion</td>
<td>52</td>
</tr>
<tr>
<td>nvmlDeviceGetInforomVersion</td>
<td>53</td>
</tr>
<tr>
<td>nvmlDeviceGetMaxClockInfo</td>
<td>53</td>
</tr>
<tr>
<td>nvmlDeviceGetMaxPcieLinkGeneration</td>
<td>54</td>
</tr>
<tr>
<td>nvmlDeviceGetMaxPcieLinkWidth</td>
<td>54</td>
</tr>
<tr>
<td>nvmlDeviceGetMemoryErrorCounter</td>
<td>55</td>
</tr>
<tr>
<td>nvmlDeviceGetMemoryInfo</td>
<td>56</td>
</tr>
<tr>
<td>nvmlDeviceGetName</td>
<td>56</td>
</tr>
<tr>
<td>nvmlDeviceGetPciInfo</td>
<td>57</td>
</tr>
<tr>
<td>nvmlDeviceGetPerformanceState</td>
<td>57</td>
</tr>
<tr>
<td>nvmlDeviceGetPersistenceMode</td>
<td>57</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerManagementDefaultLimit</td>
<td>58</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerManagementLimit</td>
<td>58</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerManagementLimitConstraints</td>
<td>59</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerManagementMode</td>
<td>60</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerState</td>
<td>60</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerUsage</td>
<td>61</td>
</tr>
<tr>
<td>nvmlDeviceGetRetiredPages</td>
<td>61</td>
</tr>
<tr>
<td>nvmlDeviceGetRetiredPagesPendingStatus</td>
<td>62</td>
</tr>
</tbody>
</table>

NVIDIA Management Library
CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.11.2.42</td>
<td>nvmlDeviceGetSerial</td>
<td>62</td>
</tr>
<tr>
<td>6.11.2.43</td>
<td>nvmlDeviceGetSupportedClocksThrottleReasons</td>
<td>63</td>
</tr>
<tr>
<td>6.11.2.44</td>
<td>nvmlDeviceGetSupportedGraphicsClocks</td>
<td>63</td>
</tr>
<tr>
<td>6.11.2.45</td>
<td>nvmlDeviceGetSupportedMemoryClocks</td>
<td>64</td>
</tr>
<tr>
<td>6.11.2.46</td>
<td>nvmlDeviceGetTemperature</td>
<td>64</td>
</tr>
<tr>
<td>6.11.2.47</td>
<td>nvmlDeviceGetTotalEccErrors</td>
<td>65</td>
</tr>
<tr>
<td>6.11.2.48</td>
<td>nvmlDeviceGetUtilizationRates</td>
<td>66</td>
</tr>
<tr>
<td>6.11.2.49</td>
<td>nvmlDeviceGetUUID</td>
<td>66</td>
</tr>
<tr>
<td>6.11.2.50</td>
<td>nvmlDeviceGetVbiosVersion</td>
<td>67</td>
</tr>
<tr>
<td>6.11.2.51</td>
<td>nvmlDeviceOnSameBoard</td>
<td>67</td>
</tr>
<tr>
<td>6.11.2.52</td>
<td>nvmlDeviceResetApplicationsClocks</td>
<td>68</td>
</tr>
<tr>
<td>6.11.2.53</td>
<td>nvmlDeviceValidateInforom</td>
<td>68</td>
</tr>
</tbody>
</table>

| 6.12 | Unit Commands | 69 |
| 6.12.1 | Detailed Description | 69 |
| 6.12.2 | Function Documentation | 69 |
| 6.12.2.1 | nvmlUnitSetLedState | 69 |

| 6.13 | Device Commands | 70 |
| 6.13.1 | Detailed Description | 70 |
| 6.13.2 | Function Documentation | 70 |
| 6.13.2.1 | nvmlDeviceClearEccErrorCounts | 70 |
| 6.13.2.2 | nvmlDeviceSetApplicationsClocks | 71 |
| 6.13.2.3 | nvmlDeviceSetComputeMode | 71 |
| 6.13.2.4 | nvmlDeviceSetDriverModel | 72 |
| 6.13.2.5 | nvmlDeviceSetEccMode | 73 |
| 6.13.2.6 | nvmlDeviceSetGpuOperationMode | 74 |
| 6.13.2.7 | nvmlDeviceSetPersistenceMode | 74 |
| 6.13.2.8 | nvmlDeviceSetPowerManagementLimit | 75 |

| 6.14 | Event Handling Methods | 76 |
| 6.14.1 | Detailed Description | 76 |
| 6.14.2 | Typedef Documentation | 76 |
| 6.14.2.1 | nvmlEventSet_t | 76 |
| 6.14.3 | Function Documentation | 76 |
| 6.14.3.1 | nvmlDeviceGetSupportedEventTypes | 76 |
| 6.14.3.2 | nvmlDeviceRegisterEvents | 77 |
| 6.14.3.3 | nvmlEventSetCreate | 78 |
| 6.14.3.4 | nvmlEventSetFree | 78 |
| 6.14.3.5 | nvmlEventSetWait | 78 |
6.15 NvmlClocksThrottleReasons .......................... 80
  6.15.1 Define Documentation .............................. 80
    6.15.1.1 nvmlClocksThrottleReasonAll ....................... 80
    6.15.1.2 nvmlClocksThrottleReasonApplicationsClocksSetting .... 80
    6.15.1.3 nvmlClocksThrottleReasonGpuIdle .................... 80
    6.15.1.4 nvmlClocksThrottleReasonHwSlowdown ............... 80
    6.15.1.5 nvmlClocksThrottleReasonNone ...................... 81
    6.15.1.6 nvmlClocksThrottleReasonSwPowerCap ............... 81
    6.15.1.7 nvmlClocksThrottleReasonUnknown ................... 81
    6.15.1.8 nvmlClocksThrottleReasonUserDefinedClocks ......... 81

7 Data Structure Documentation 83
  7.1 nvmlAccountingStats_t Struct Reference ............... 83
      7.1.1 Detailed Description .............................. 83
  7.2 nvmlEccErrorCounts_t Struct Reference .................. 84
      7.2.1 Detailed Description .............................. 84
  7.3 nvmlEventData_t Struct Reference ....................... 85
      7.3.1 Detailed Description .............................. 85
  7.4 nvmlHwbcEntry_t Struct Reference ....................... 86
      7.4.1 Detailed Description .............................. 86
  7.5 nvmlLedState_t Struct Reference ....................... 87
      7.5.1 Detailed Description .............................. 87
  7.6 nvmlMemory_t Struct Reference ......................... 88
      7.6.1 Detailed Description .............................. 88
  7.7 nvmlPciInfo_t Struct Reference ........................ 89
      7.7.1 Detailed Description .............................. 89
  7.8 nvmlProcessInfo_t Struct Reference ..................... 90
      7.8.1 Detailed Description .............................. 90
  7.9 nvmlPSUInfo_t Struct Reference ......................... 91
      7.9.1 Detailed Description .............................. 91
  7.10 nvmlUnitFanInfo_t Struct Reference .................... 92
       7.10.1 Detailed Description ............................ 92
  7.11 nvmlUnitFanSpeeds_t Struct Reference ................... 93
       7.11.1 Detailed Description ............................ 93
  7.12 nvmlUnitInfo_t Struct Reference ....................... 94
       7.12.1 Detailed Description ............................ 94
  7.13 nvmlUtilization_t Struct Reference .................... 95
Chapter 1

Known issues in the current version of NVML library

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

- On Linux when X Server is running `nvmlDeviceGetComputeRunningProcesses` may return a `nvmlProcessInfo_usedGpuMemory` value that is larger than the actual value. This will be fixed in a future release.

- On Linux GPU Reset can’t be triggered when there is pending GPU Operation Mode (GOM) change

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

- **Accounting Statistics** supports only one process per GPU at a time (CUDA proxy server counts as one process).

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

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

- `nvmlDeviceGetAccountingStats` memory utilization information is disabled on all GPUs
Chapter 2

Change log of NVML library
This chapter list changes in API and bug fixes that were introduced to the library

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

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

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

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

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

- Added nvmlDeviceGetGpuOperationMode and nvmlDeviceSetGpuOperationMode

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

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

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

2.4 Changes between NVML v2.285 and v3.295

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

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

2.5 Changes between NVML v1.0 and v2.285

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

• Added possibility to query separately current and pending driver model with `nvmlDeviceGetDriverModel`
• Added API `nvmlDeviceGetVbiosVersion` function to report VBIOS version.
• Added `pciSubSystemId` to `nvmlPciInfo_t` struct
• Added API `nvmlErrorString` function to convert error code to string
• Updated docs to indicate we support M2075 and C2075
• Added API `nvmlSystemGetHicVersion` function to report HIC firmware version
• Added NVML versioning support
  – Functions that changed API and/or size of structs have appended versioning suffix (e.g. nvmlDeviceGetPciInfo_v2). Appropriate C defines have been added that map old function names to the newer version of the function

• Added support for concurrent library usage by multiple libraries

• Added API nvmlDeviceGetMaxClockInfo function for reporting device’s clock limits

• Added new error code NVML_ERROR_DRIVER_NOT_LOADED used by nvmlInit

• Extended nvmlPciInfo_t struct with new field: sub system id

• Added NVML support on Windows guest account

• Changed format of pciBusId string (to XXXX:XX:XX.X) of nvmlPciInfo_t

• Parsing of busId in nvmlDeviceGetHandleByPciBusId is less restrictive. You can pass 0:2:0.0 or 0000:02:00 and other variations

• Added API for events waiting for GPU events (Linux only) see docs of Event Handling Methods

• Added API nvmlDeviceGetComputeRunningProcesses and nvmlSystemGetProcessName functions for looking up currently running compute applications

• Deprecated nvmlDeviceGetPowerState in favor of nvmlDeviceGetPerformanceState.
Chapter 3

Deprecated List
Class `nvmlEccErrorCounts_t` Different GPU families can have different memory error counters See `nvmlDeviceGetMemoryErrorCounter`

Global `NVML_DOUBLE_BIT_ECC` Mapped to `NVML_MEMORY_ERROR_TYPE_UNCORRECTED`

Global `NVML_SINGLE_BIT_ECC` Mapped to `NVML_MEMORY_ERROR_TYPE_CORRECTED`

Global `nvmlEccBitType_t` See `nvmlMemoryErrorType_t` for a more flexible type

Global `nvmlDeviceGetDetailedEccErrors` This API supports only a fixed set of ECC error locations On different GPU architectures different locations are supported See `nvmlDeviceGetMemoryErrorCounter`

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

Global `nvmlClocksThrottleReasonUserDefinedClocks` Renamed to `nvmlClocksThrottleReasonApplicationClocksSetting` as the name describes the situation more accurately.
Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

- Device Structs .......................................................... 13
- Device Enums ........................................................... 14
- Unit Structs ............................................................... 21
- Accounting Statistics .................................................. 24
- Initialization and Cleanup ............................................. 28
- Error reporting ......................................................... 30
- Constants ................................................................. 31
- System Queries .......................................................... 32
- Unit Queries .............................................................. 34
- Device Queries ......................................................... 38
- Unit Commands ......................................................... 69
- Device Commands ..................................................... 70
- Event Handling Methods .............................................. 76
- Event Types ............................................................. 22
- NvmlClocksThrottleReasons ........................................ 80
Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

- `nvmlAccountingStats_t` ................................................................. 83
- `nvmlEccErrorCounts_t` ................................................................. 84
- `nvmlEventData_t` ................................................................. 85
- `nvmlHwbcEntry_t` ................................................................. 86
- `nvmlLedState_t` ................................................................. 87
- `nvmlMemory_t` ................................................................. 88
- `nvmlPciInfo_t` ................................................................. 89
- `nvmlProcessInfo_t` ................................................................. 90
- `nvmlPSUInfo_t` ................................................................. 91
- `nvmlUnitFanInfo_t` ................................................................. 92
- `nvmlUnitFanSpeeds_t` ................................................................. 93
- `nvmlUnitInfo_t` ................................................................. 94
- `nvmlUtilization_t` ................................................................. 95
Chapter 6

Module Documentation

6.1 Device Structs

Data Structures

- struct nvmlPciInfo_t
- struct nvmlEccErrorCounts_t
- struct nvmlUtilization_t
- struct nvmlMemory_t
- struct nvmlProcessInfo_t

Defines

- #define NVML_VALUE_NOT_AVAILABLE (-1)
- #define NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE 16

6.1.1 Define Documentation

6.1.1.1 #define NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE 16

Buffer size guaranteed to be large enough for pci bus id

6.1.1.2 #define NVML_VALUE_NOT_AVAILABLE (-1)

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

Defines

- `#define nvmlFlagDefault 0x00`
  
  Generic flag used to specify the default behavior of some functions. See description of particular functions for details.

- `#define nvmlFlagForce 0x01`
  
  Generic flag used to force some behavior. See description of particular functions for details.

- `#define nvmlEccBitType_t nvmlMemoryErrorType_t`
- `#define NVML_SINGLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_CORRECTED`
- `#define NVML_DOUBLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_UNCORRECTED`

Enumerations

- `enum nvmlEnableState_t {
    NVML_FEATURE_DISABLED = 0,
    NVML_FEATURE_ENABLED = 1
  }`
- `enum nvmlTemperatureSensors_t { NVML_TEMPERATURE_GPU = 0 }`
- `enum nvmlComputeMode_t {
    NVML_COMPUTEMODE_DEFAULT = 0,
    NVML_COMPUTEMODE_EXCLUSIVE_THREAD = 1,
    NVML_COMPUTEMODE_PROHIBITED = 2,
    NVML_COMPUTEMODE_EXCLUSIVE_PROCESS = 3
  }`
- `enum nvmlMemoryErrorType_t {
    NVML_MEMORY_ERROR_TYPE_CORRECTED = 0,
    NVML_MEMORY_ERROR_TYPE_UNCORRECTED = 1
  }`
- `enum nvmlEccCounterType_t {
    NVML_VOLATILE_ECC = 0,
    NVML_AGGREGATE_ECC = 1
  }`
- `enum nvmlClockType_t {
    NVML_CLOCK_GRAPHICS = 0,
    NVML_CLOCK_SM = 1,
    NVML_CLOCK_MEM = 2
  }`
- `enum nvmlDriverModel_t {
    NVML_DRIVER_WDDM = 0,
    NVML_DRIVER_WDM = 1
  }`
- `enum nvmlPstates_t {
    NVML_PSTATE_0 = 0,
    NVML_PSTATE_1 = 1,
    NVML_PSTATE_2 = 2,
    NVML_PSTATE_3 = 3
  }`
6.2 Device Enums

```
NVML_PSTATE_4 = 4,
NVML_PSTATE_5 = 5,
NVML_PSTATE_6 = 6,
NVML_PSTATE_7 = 7,
NVML_PSTATE_8 = 8,
NVML_PSTATE_9 = 9,
NVML_PSTATE_10 = 10,
NVML_PSTATE_11 = 11,
NVML_PSTATE_12 = 12,
NVML_PSTATE_13 = 13,
NVML_PSTATE_14 = 14,
NVML_PSTATE_15 = 15,
NVML_PSTATE_UNKNOWN = 32 } 

• enum nvmlGpuOperationMode_t {
    NVML_GOM_ALL_ON = 0,
    NVML_GOM_COMPUTE = 1,
    NVML_GOM_LOW_DP = 2 }

• enum nvmlInforomObject_t {
    NVML_INFOROM_OEM = 0,
    NVML_INFOROM_ECC = 1,
    NVML_INFOROM_POWER = 2,
    NVML_INFOROM_COUNT }

• enum nvmlReturn_t {
    NVML_SUCCESS = 0,
    NVML_ERROR_UNINITIALIZED = 1,
    NVML_ERROR_INVALID_ARGUMENT = 2,
    NVML_ERROR_NOT_SUPPORTED = 3,
    NVML_ERROR_NO_PERMISSION = 4,
    NVML_ERROR_ALREADY_INITIALIZED = 5,
    NVML_ERROR_NOT_FOUND = 6,
    NVML_ERROR_INSUFFICIENT_SIZE = 7,
    NVML_ERROR_INSUFFICIENT_POWER = 8,
    NVML_ERROR_DRIVER_NOT_LOADED = 9,
    NVML_ERROR_TIMEOUT = 10,
    NVML_ERROR_IRQ_ISSUE = 11,
    NVML_ERROR_LIBRARY_NOT_FOUND = 12,
    NVML_ERROR_FUNCTION_NOT_FOUND = 13,
    NVML_ERROR_CORRUPTED_INFOROM = 14,
    NVML_ERROR_GPU_IS_LOST = 15,
    NVML_ERROR_UNKNOWN = 999 }
```
enum nvmlMemoryLocation_t {
    NVML_MEMORY_LOCATION_L1_CACHE = 0,
    NVML_MEMORY_LOCATION_L2_CACHE = 1,
    NVML_MEMORY_LOCATION_DEVICE_MEMORY = 2,
    NVML_MEMORY_LOCATION_REGISTER_FILE = 3,
    NVML_MEMORY_LOCATION_TEXTURE_MEMORY = 4,
    NVML_MEMORY_LOCATION_COUNT
}

enum nvmlPageRetirementCause_t {
    NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS = 0,
    NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR = 1
}

6.2.1 Define Documentation

6.2.1.1 #define NVML_DOUBLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_UNCORRECTED
Double bit ECC errors
Deprecated
Mapped to NVML_MEMORY_ERROR_TYPE_UNCORRECTED

6.2.1.2 #define NVML_SINGLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_CORRECTED
Single bit ECC errors
Deprecated
Mapped to NVML_MEMORY_ERROR_TYPE_CORRECTED

6.2.1.3 #define nvmlEccBitType_t nvmlMemoryErrorType_t
ECC bit types.
Deprecated
See nvmlMemoryErrorType_t for a more flexible type

6.2.2 Enumeration Type Documentation

6.2.2.1 enum nvmlClockType_t
Clock types.
All speeds are in Mhz.
Enumerator:

    NVML_CLOCK_GRAPHICS  Graphics clock domain.
    NVML_CLOCK_SM       SM clock domain.
    NVML_CLOCK_MEM      Memory clock domain.
6.2.2.2  enum nvmlComputeMode_t

Compute mode.

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

**Enumerator:**

- **NVML_COMPUTEMODE_DEFAULT**  Default compute mode – multiple contexts per device.
- **NVML_COMPUTEMODE_EXCLUSIVE_THREAD**  Compute-exclusive-thread mode – only one context per device, usable from one thread at a time.
- **NVML_COMPUTEMODE_PROHIBITED**  Compute-prohibited mode – no contexts per device.
- **NVML_COMPUTEMODE_EXCLUSIVE_PROCESS**  Compute-exclusive-process mode – only one context per device, usable from multiple threads at a time.

6.2.2.3  enum nvmlDriverModel_t

Driver models.

Windows only.

**Enumerator:**

- **NVML_DRIVER_WDDM**  WDDM driver model – GPU treated as a display device.
- **NVML_DRIVER_WDM**  WDM (TCC) model (recommended) – GPU treated as a generic device.

6.2.2.4  enum nvmlEccCounterType_t

ECC counter types.

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

**Enumerator:**

- **NVML_VOLATILE_ECC**  Volatile counts are reset each time the driver loads.
- **NVML_AGGREGATE_ECC**  Aggregate counts persist across reboots (i.e. for the lifetime of the device).
- **NVML_ECC_COUNTER_TYPE_COUNT**  Count of memory counter types.

6.2.2.5  enum nvmlEnableState_t

Generic enable/disable enum.

**Enumerator:**

- **NVML_FEATURE_DISABLED**  Feature disabled.
- **NVML_FEATURE_ENABLED**  Feature enabled.
6.2.2.6  enum nvmlGpuOperationMode_t

GPU Operation Mode
GOM allows to reduce power usage and optimize GPU throughput by disabling GPU features.
Each GOM is designed to meet specific user needs.

Enumerator:

    NVML_GOM_ALL_ON  Everything is enabled and running at full speed.
    NVML_GOM_COMPUTE Designed for running only compute tasks. Graphics operations < are not allowed.
    NVML_GOM_LOW_DP  Designed for running graphics applications that don’t require < high bandwidth double precision.

6.2.2.7  enum nvmlInforomObject_t

Available infoROM objects.

Enumerator:

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

6.2.2.8  enum nvmlMemoryErrorType_t

Memory error types

Enumerator:

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

6.2.2.9  enum nvmlMemoryLocation_t

Memory locations
See nvmlDeviceGetMemoryErrorCounter

Enumerator:

    NVML_MEMORY_LOCATION_L1_CACHE GPU L1 Cache.
    NVML_MEMORY_LOCATION_L2_CACHE GPU L2 Cache.
    NVML_MEMORY_LOCATION_DEVICE_MEMORY GPU Device Memory.
    NVML_MEMORY_LOCATION_REGISTER_FILE GPU Register File.
    NVML_MEMORY_LOCATION_TEXTURE_MEMORY GPU Texture Memory.
    NVML_MEMORY_LOCATION_COUNT This counts the number of memory locations the driver knows about.
6.2 Device Enums

6.2.2.10 enum nvmlPageRetirementCause_t

Causes for page retirement

Enumerator:

- **NVML_PAGE_RETIREMENT_CAUSE_MULTIPLE_SINGLE_BIT_ECC_ERRORS** Page was retired due to multiple single bit ECC error.
- **NVML_PAGE_RETIREMENT_CAUSE_DOUBLE_BIT_ECC_ERROR** Page was retired due to double bit ECC error.

6.2.2.11 enum nvmlPstates_t

Allowed PStates.

Enumerator:

- **NVML_PSTATE_0** Performance state 0 – Maximum Performance.
- **NVML_PSTATE_1** Performance state 1.
- **NVML_PSTATE_2** Performance state 2.
- **NVML_PSTATE_3** Performance state 3.
- **NVML_PSTATE_4** Performance state 4.
- **NVML_PSTATE_5** Performance state 5.
- **NVML_PSTATE_6** Performance state 6.
- **NVML_PSTATE_7** Performance state 7.
- **NVML_PSTATE_8** Performance state 8.
- **NVML_PSTATE_9** Performance state 9.
- **NVML_PSTATE_10** Performance state 10.
- **NVML_PSTATE_11** Performance state 11.
- **NVML_PSTATE_12** Performance state 12.
- **NVML_PSTATE_13** Performance state 13.
- **NVML_PSTATE_14** Performance state 14.
- **NVML_PSTATE_15** Performance state 15 – Minimum Performance.
- **NVML_PSTATE_UNKNOWN** Unknown performance state.

6.2.2.12 enum nvmlReturn_t

Return values for NVML API calls.

Enumerator:

- **NVML_SUCCESS** The operation was successful.
- **NVML_ERROR_UNINITIALIZED** NVML was not first initialized with `nvmlInit()`.
- **NVML_ERROR_INVALID_ARGUMENT** A supplied argument is invalid.
- **NVML_ERROR_NOT_SUPPORTED** The requested operation is not available on target device.
- **NVML_ERROR_NO_PERMISSION** The current user does not have permission for operation.
**NVML_ERROR_ALREADY_INITIALIZED**  Deprecated: Multiple initializations are now allowed through ref counting.

**NVML_ERROR_NOT_FOUND**  A query to find an object was unsuccessful.

**NVML_ERROR_INSUFFICIENT_SIZE**  An input argument is not large enough.

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

**NVML_ERROR_DRIVER_NOT_LOADED**  NVIDIA driver is not loaded.

**NVML_ERROR_TIMEOUT**  User provided timeout passed.

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

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

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

**NVML_ERROR_CORRUPTED_INFOROM**  infoROM is corrupted

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

**NVML_ERROR_UNKNOWN**  An internal driver error occurred.

### 6.2.2.13 enum nvmlTemperatureSensors_t

Temperature sensors.

**Enumerator:**

- **NVML_TEMPERATURE_GPU**  Temperature sensor for the GPU die.
6.3 Unit Structs

Data Structures

- struct nvmlHwbcEntry_t
- struct nvmlLedState_t
- struct nvmlUnitInfo_t
- struct nvmlPSUInfo_t
- struct nvmlUnitFanInfo_t
- struct nvmlUnitFanSpeeds_t

Enumerations

- enum nvmlFanState_t {
  NVML_FAN_NORMAL = 0,
  NVML_FAN_FAILED = 1
}
- enum nvmlLedColor_t {
  NVML_LED_COLOR_GREEN = 0,
  NVML_LED_COLOR_AMBER = 1
}

6.3.1 Enumeration Type Documentation

6.3.1.1 enum nvmlFanState_t

Fan state enum.

Enumerator:

  NVML_FAN_NORMAL  Fan is working properly.
  NVML_FAN_FAILED  Fan has failed.

6.3.1.2 enum nvmlLedColor_t

Led color enum.

Enumerator:

  NVML_LED_COLOR_GREEN  GREEN, indicates good health.
  NVML_LED_COLOR_AMBER  AMBER, indicates problem.
6.4 Event Types

Defines

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

- #define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL
  Event about double bit ECC errors.

- #define nvmlEventTypePState 0x0000000000000004LL
  Event about PState changes.

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

- #define nvmlEventTypeClock 0x0000000000000010LL
  Event about clock changes.

- #define nvmlEventTypeNone 0x0000000000000000LL
  Mask with no events.

- #define nvmlEventTypeAll
  Mask of all events.

6.4.1 Detailed Description

Event Types which user can be notified about. See description of particular functions for details.
See `nvmlDeviceRegisterEvents` and `nvmlDeviceGetSupportedEventTypes` to check which devices support each event.
Types can be combined with bitwise or operator `|` when passed to `nvmlDeviceRegisterEvents`

6.4.2 Define Documentation

6.4.2.1 #define nvmlEventTypeClock 0x0000000000000010LL

Kepler only

6.4.2.2 #define nvmlEventTypeDoubleBitEccError 0x0000000000000002LL

Note:

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

6.4.2.3  
#define nvmlEventTypePState 0x0000000000000004LL

Note:

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

6.4.2.4  
#define nvmlEventTypeSingleBitEccError 0x0000000000000001LL

Note:

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

Data Structures

- struct nvmlAccountingStats_t

Functions

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

6.5.1 Detailed Description

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

Note:

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

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

6.5.2 Function Documentation

6.5.2.1 nvmlReturn_t DECLDIR nvmlDeviceClearAccountingPids (nvmlDevice_t device)

Clears accounting information about all processes that have already terminated.

For Tesla ™ and Quadro ® products from the Kepler family. Requires root/admin permissions.

See nvmlDeviceGetAccountingMode See nvmlDeviceGetAccountingStats See nvmlDeviceSetAccountingMode

Parameters:

device The identifier of the target device

Returns:

- NVML_SUCCESS if accounting information has been cleared
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if device are invalid
- NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
- NVML_ERROR_NO_PERMISSION if the user doesn’t have permission to perform this operation
- NVML_ERROR_UNKNOWN on any unexpected error
6.5.2.2 nvmlReturn_t DECLDIR nvmlDeviceGetAccountingBufferSize (nvmlDevice_t device, unsigned int *bufferSize)

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

For Tesla™ and Quadro® products from the Kepler family.

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

Parameters:

device  The identifier of the target device

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

Returns:

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

See also:

nvmlDeviceGetAccountingStats
nvmlDeviceGetAccountingPids

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

Queries the state of per process accounting mode.

For Tesla™ and Quadro® products from the Kepler family.

See nvmlDeviceGetAccountingStats for more details. See nvmlDeviceSetAccountingMode

Parameters:

device  The identifier of the target device

mode  Reference in which to return the current accounting mode

Returns:

• NVML_SUCCESS if the mode has been successfully retrieved
• NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
• NVML_ERROR_INVALID_ARGUMENT if device is invalid or mode are NULL
• NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
• NVML_ERROR_UNKNOWN on any unexpected error
6.5.2.4  `nvmlReturn_t DECLDIR nvmlDeviceGetAccountingPids (nvmlDevice_t device, unsigned int * count, unsigned int * pids)`

Queries list of processes that can be queried for accounting stats.

For Tesla ™ and Quadro ® products from the Kepler family.

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

For more details see `nvmlDeviceGetAccountingStats`.

**Note:**

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

**Parameters:**

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

**Returns:**

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

See also:

- `nvmlDeviceGetAccountingBufferSize`

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

Queries process’s accounting stats.

For Tesla ™ and Quadro ® products from the Kepler family.

Accounting stats capture GPU utilization and other statistics across the lifetime of a process. Accounting stats can be queried during life time of the process and after its termination. Accounting stats are kept in a circular buffer, newly created processes overwrite information about old processes.

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

**Note:**

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

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

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

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

Parameters:

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

Returns:

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

See also:

- `nvmlDeviceGetAccountingBufferSize`

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

Enables or disables per process accounting.

For Tesla™ and Quadro® products from the Kepler family. Requires root/admin permissions.

Note:

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

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

Disabling accounting clears all accounting pids information.

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

Parameters:

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

Returns:

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

Functions

- `nvmlReturn_t DECLDIR nvmlInit (void)`
- `nvmlReturn_t DECLDIR nvmlShutdown (void)`

6.6.1 Detailed Description

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

6.6.2 Function Documentation

6.6.2.1 `nvmlReturn_t DECLDIR nvmlInit (void)`

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

Note:

In NVML 5.319 new `nvmlInit_v2` has replaced `nvmlInit_v1` (default in NVML 4.304 and older) that did initialize all GPU devices in the system.

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

Note:

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

For all products.

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

Returns:

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

6.6.2.2 `nvmlReturn_t DECLDIR nvmlShutdown (void)`

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

For all products.

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

Returns:

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

Functions

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

6.7.1 Detailed Description

This chapter describes helper functions for error reporting routines.

6.7.2 Function Documentation

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

Helper method for converting NVML error codes into readable strings.

For all products

Parameters:

  - `result` NVML error code to convert

Returns:

  String representation of the error.
6.8 Constants

Defines

• #define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16
• #define NVML_DEVICE_UUID_BUFFER_SIZE 80
• #define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80
• #define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80
• #define NVML_DEVICE_NAME_BUFFER_SIZE 64
• #define NVML_DEVICE_SERIAL_BUFFER_SIZE 30
• #define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32

6.8.1 Define Documentation

6.8.1.1 #define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16

Buffer size guaranteed to be large enough for nvmlDeviceGetInforomVersion and nvmlDeviceGetInforomImageVersion

6.8.1.2 #define NVML_DEVICE_NAME_BUFFER_SIZE 64

Buffer size guaranteed to be large enough for nvmlDeviceGetName

6.8.1.3 #define NVML_DEVICE_SERIAL_BUFFER_SIZE 30

Buffer size guaranteed to be large enough for nvmlDeviceGetSerial

6.8.1.4 #define NVML_DEVICE_UUID_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlDeviceGetUUID

6.8.1.5 #define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32

Buffer size guaranteed to be large enough for nvmlDeviceGetVbiosVersion

6.8.1.6 #define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlSystemGetDriverVersion

6.8.1.7 #define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80

Buffer size guaranteed to be large enough for nvmlSystemGetNVMLVersion
6.9 System Queries

Functions

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

6.9.1 Detailed Description

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

6.9.2 Function Documentation

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

Retrieves the version of the system’s graphics driver.

For all products.

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

Parameters:

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

Returns:

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

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

Retrieves the version of the NVML library.

For all products.

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

Parameters:

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

Returns:

- `NVML_SUCCESS` if `version` has been set
6.9 System Queries

- NVML_ERROR_INVALID_ARGUMENT if `version` is NULL
- NVML_ERROR_INSUFFICIENT_SIZE if `length` is too small

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

Gets name of the process with provided process id
For all products.
Returned process name is cropped to provided length. name string is encoded in ANSI.

Parameters:

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

Returns:

- NVML_SUCCESS if `name` has been set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if `name` is NULL
- 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
6.10 Unit Queries

Functions

- \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetCount} (unsigned int *unitCount)
- \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetHandleByIndex} (unsigned int index, nvmlUnit\_t *unit)
- \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetUnitInfo} (nvmlUnit\_t unit, nvmlUnitInfo\_t *info)
- \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetLedState} (nvmlUnit\_t unit, nvmlLedState\_t *state)
- \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetPsuInfo} (nvmlUnit\_t unit, nvmlPsuInfo\_t *psu)
- \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetTemperature} (nvmlUnit\_t unit, unsigned int type, unsigned int *temp)
- \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetFanSpeedInfo} (nvmlUnit\_t unit, nvmlUnitFanSpeeds\_t *fanSpeeds)
- \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetDevices} (nvmlUnit\_t unit, unsigned int *deviceCount, nvmlDevice\_t *devices)
- \texttt{nvmlReturn\_t DECLDIR nvmlSystemGetHicVersion} (unsigned int *hwbcCount, nvmlHwbcEntry\_t *hwbcEntries)

6.10.1 Detailed Description

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

6.10.2 Function Documentation

6.10.2.1 \texttt{nvmlReturn\_t DECLDIR nvmlSystemGetHicVersion} (unsigned int *hwbcCount, nvmlHwbcEntry\_t *hwbcEntries)

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

For S-class products.

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

Parameters:

- \texttt{hwbcCount}  Size of hwbcEntries array
- \texttt{hwbcEntries}  Array holding information about hwbc

Returns:

- NVML\_SUCCESS if \texttt{hwbcCount} and \texttt{hwbcEntries} have been populated
- NVML\_ERROR\_UNINITIALIZED if the library has not been successfully initialized
- NVML\_ERROR\_INVALID\_ARGUMENT if either \texttt{hwbcCount} or \texttt{hwbcEntries} is NULL
- NVML\_ERROR\_INSUFFICIENT\_SIZE if \texttt{hwbcCount} indicates that the \texttt{hwbcEntries} array is too small

6.10.2.2 \texttt{nvmlReturn\_t DECLDIR nvmlUnitGetCount} (unsigned int *unitCount)

Retrieves the number of units in the system.

For S-class products.
6.10 Unit Queries

Parameters:

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

Returns:

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

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

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

For S-class products.

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

Parameters:

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

Returns:

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

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

Retrieves the fan speed readings for the unit.

For S-class products.

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

Parameters:

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

Returns:

- NVML_SUCCESS if `fanSpeeds` has been populated
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_INVALID_ARGUMENT if `unit` is invalid or `fanSpeeds` is NULL
- NVML_ERROR_NOT_SUPPORTED if this is not an S-class product
- NVML_ERROR_UNKNOWN on any unexpected error
6.10.2.5 nvmlReturn_t DECLDIR nvmlUnitGetHandleByIndex (unsigned int index, nvmlUnit_t *unit)

Acquire the handle for a particular unit, based on its index.
For S-class products.
Valid indices are derived from the unitCount returned by nvmlUnitGetCount(). For example, if unitCount is 2 the valid indices are 0 and 1, corresponding to UNIT 0 and UNIT 1.
The order in which NVML enumerates units has no guarantees of consistency between reboots.

Parameters:
- **index** The index of the target unit, \( \geq 0 \) and \( < \) unitCount
- **unit** Reference in which to return the unit handle

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

6.10.2.6 nvmlReturn_t DECLDIR nvmlUnitGetLedState (nvmlUnit_t unit, nvmlLedState_t *state)

Retrieves the LED state associated with this unit.
For S-class products.
See nvmlLedState_t for details on allowed states.

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

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

See also:
- nvmlUnitSetLedState()

6.10.2.7 nvmlReturn_t DECLDIR nvmlUnitGetPsuInfo (nvmlUnit_t unit, nvmlPSUInfo_t *psu)

Retrieves the PSU stats for the unit.
For S-class products.
See nvmlPSUInfo_t for details on available PSU info.
6.10 Unit Queries

Parameters:

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

Returns:

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

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

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

For S-class products.

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

Parameters:

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

Returns:

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

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

Retrieves the static information associated with a unit.

For S-class products.

See `nvmlUnitInfo_t` for details on available unit info.

Parameters:

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

Returns:

- **NVML_SUCCESS** if *info* has been populated
- **NVML_ERROR_UNINITIALIZED** if the library has not been successfully initialized
- **NVML_ERROR_INVALID_ARGUMENT** if *unit* is invalid or *info* is NULL
6.11 Device Queries

Functions

- `nvmlReturn_t DECLDIR nvmlDeviceGetCount (unsigned int *deviceCount)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetHandleByIndex (unsigned int index, nvmlDevice_t *device)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetHandleBySerial (const char *serial, nvmlDevice_t *device)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetHandleByUUID (const char *uuid, nvmlDevice_t *device)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetHandleByPciBusId (const char *pciBusId, nvmlDevice_t *device)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetName (nvmlDevice_t device, char *name, unsigned int length)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetIndex (nvmlDevice_t device, unsigned int *index)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetSerial (nvmlDevice_t device, char *serial, unsigned int length)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetUUID (nvmlDevice_t device, char *uuid, unsigned int length)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetInforomVersion (nvmlDevice_t device, nvmlInforomObject_t object, char *version, unsigned int length)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetInforomImageVersion (nvmlDevice_t device, char *version, unsigned int length)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetInforomConfigurationChecksum (nvmlDevice_t device, unsigned int *checksum)`
- `nvmlReturn_t DECLDIR nvmlDeviceValidateInforom (nvmlDevice_t device)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetDisplayMode (nvmlDevice_t device, nvmlEnableState_t *display)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetDisplayActive (nvmlDevice_t device, nvmlEnableState_t *isActive)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t *mode)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetPciInfo (nvmlDevice_t device, nvmlPciInfo_t *pci)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkGeneration (nvmlDevice_t device, unsigned int *maxLinkGen)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkWidth (nvmlDevice_t device, unsigned int *maxLinkWidth)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkGeneration (nvmlDevice_t device, unsigned int *currLinkGen)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkWidth (nvmlDevice_t device, unsigned int *currLinkWidth)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetMaxClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetApplicationsClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int *clockMHz)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetDefaultApplicationsClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int *clockMHz)`
- `nvmlReturn_t DECLDIR nvmlDeviceResetApplicationsClocks (nvmlDevice_t device)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetSupportedMemoryClocks (nvmlDevice_t device, unsigned int *count, unsigned int *clocksMHz)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetSupportedGraphicsClocks (nvmlDevice_t device, unsigned int memoryClockMHz, unsigned int *count, unsigned int *clocksMHz)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetFanSpeed (nvmlDevice_t device, unsigned int *speed)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetTemperature (nvmlDevice_t device, nvmlTemperatureSensors_t sensorType, unsigned int *temp)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetPerformanceState (nvmlDevice_t device, nvmlPstates_t *pState)`
- `nvmlReturn_t DECLDIR nvmlDeviceGetCurrentClocksThrottleReasons (nvmlDevice_t device, unsigned long long *clocksThrottleReasons)`
• nvmlReturn_t DECLDIR nvmlDeviceGetSupportedClocksThrottleReasons (nvmlDevice_t device, unsigned long long *supportedClocksThrottleReasons)
• nvmlReturn_t DECLDIR nvmlDeviceGetPowerState (nvmlDevice_t device, nvmlPstates_t *pState)
• nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementMode (nvmlDevice_t device, nvmlEnableState_t *mode)
• nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementLimit (nvmlDevice_t device, unsigned int *limit)
• nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementLimitConstraints (nvmlDevice_t device, unsigned int *minLimit, unsigned int *maxLimit)
• nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementDefaultLimit (nvmlDevice_t device, unsigned int *defaultLimit)
• nvmlReturn_t DECLDIR nvmlDeviceGetPowerUsage (nvmlDevice_t device, unsigned int *power)
• nvmlReturn_t DECLDIR nvmlDeviceGetGpuOperationMode (nvmlDevice_t device, nvmlGpuOperationMode_t *current, nvmlGpuOperationMode_t *pending)
• nvmlReturn_t DECLDIR nvmlDeviceGetMemoryInfo (nvmlDevice_t device, nvmlMemory_t *memory)
• nvmlReturn_t DECLDIR nvmlDeviceGetComputeMode (nvmlDevice_t device, nvmlComputeMode_t *mode)
• nvmlReturn_t DECLDIR nvmlDeviceGetEccMode (nvmlDevice_t device, nvmlEnableState_t *current, nvmlEnableState_t *pending)
• nvmlReturn_t DECLDIR nvmlDeviceGetTotalEccErrors (nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, unsigned long long *eccCounts)
• nvmlReturn_t DECLDIR nvmlDeviceGetDetailedEccErrors (nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, nvmlEccErrorCounts_t *eccCounts)
• nvmlReturn_t DECLDIR nvmlDeviceGetMemoryErrorCounter (nvmlDevice_t device, nvmlMemoryErrorType_t errorType, nvmlEccCounterType_t counterType, nvmlMemoryLocation_t locationType, unsigned long long *count)
• nvmlReturn_t DECLDIR nvmlDeviceGetUtilizationRates (nvmlDevice_t device, nvmlUtilization_t *utilization)
• nvmlReturn_t DECLDIR nvmlDeviceGetDriverModel (nvmlDevice_t device, nvmlDriverModel_t *current, nvmlDriverModel_t *pending)
• nvmlReturn_t DECLDIR nvmlDeviceGetVbiosVersion (nvmlDevice_t device, char *version, unsigned int length)
• nvmlReturn_t DECLDIR nvmlDeviceGetComputeRunningProcesses (nvmlDevice_t device, unsigned int *infoCount, nvmlProcessInfo_t *infos)
• nvmlReturn_t DECLDIR nvmlDeviceOnSameBoard (nvmlDevice_t device1, nvmlDevice_t device2, int *onSameBoard)
• nvmlReturn_t DECLDIR nvmlDeviceGetRetiredPages (nvmlDevice_t device, nvmlPageRetirementCause_t cause, unsigned int *pageCount, unsigned long long *addresses)
• nvmlReturn_t DECLDIR nvmlDeviceGetRetiredPagesPendingStatus (nvmlDevice_t device, nvmlEnableState_t *isPending)

6.11.1 Detailed Description

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

6.11.2 Function Documentation

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

Retrieves the current setting of a clock that applications will use unless an overspec situation occurs. Can be changed using nvmlDeviceSetApplicationsClocks.
For Tesla™ products from the Kepler family.

**Parameters:**

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

**Returns:**

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

---

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

Retrieves the current clock speeds for the device.

For Tesla™ and Quadro ® products from the Fermi and Kepler families.

See `nvmlClockType_t` for details on available clock information.

**Parameters:**

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

**Returns:**

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

---

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

Retrieves the current compute mode for the device.

For all CUDA-capable products.

See `nvmlComputeMode_t` for details on allowed compute modes.
### 6.11 Device Queries

#### Parameters:

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

#### Returns:

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

#### See also:

- `nvmlDeviceSetComputeMode`

---

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

Get information about processes with a compute context on a device

For Tesla™ and Quadro @products from the Fermi and Kepler families.

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

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

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

#### Parameters:

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

#### Returns:

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

#### See also:

- `nvmlSystemGetProcessName`
6.11.2.5  \texttt{nvmlReturn_t DECLDIR nvmlDeviceGetCount (unsigned int * deviceCount)}

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

Note: New \texttt{nvmlDeviceGetCount\_v2} (default in NVML 5.319) returns count of all devices in the system even if \texttt{nvmlDeviceGetHandleByIndex\_v2} returns \texttt{NVML\_ERROR\_NO\_PERMISSION} for such device. Update your code to handle this error, or use NVML 4.304 or older \texttt{nvml} header file. For backward binary compatibility reasons \_\_v1 version of the API is still present in the shared library. Old \_\_v1 version of \texttt{nvmlDeviceGetCount} doesn’t count devices that NVML has no permission to talk to.

Parameters:

- \texttt{deviceCount}  Reference in which to return the number of accessible devices

Returns:

- \texttt{NVML\_SUCCESS} if \texttt{deviceCount} has been set
- \texttt{NVML\_ERROR\_UNINITIALIZED} if the library has not been successfully initialized
- \texttt{NVML\_ERROR\_INVALID\_ARGUMENT} if \texttt{deviceCount} is NULL
- \texttt{NVML\_ERROR\_UNKNOWN} on any unexpected error

6.11.2.6  \texttt{nvmlReturn_t DECLDIR nvmlDeviceGetCurrentClocksThrottleReasons (nvmlDevice_t device, unsigned long long * clocksThrottleReasons)}

Retrieves current clocks throttling reasons.
For Tesla ™ products from Kepler family.

Note:

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

Parameters:

- \texttt{device}  The identifier of the target device
- \texttt{clocksThrottleReasons}  Reference in which to return bitmask of active clocks throttle reasons

Returns:

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

See also:

- \texttt{NvmlClocksThrottleReasons}
- \texttt{nvmlDeviceGetSupportedClocksThrottleReasons}
6.11.2.7 nvmlReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkGeneration (nvmlDevice_t device, unsigned int *currLinkGen)

Retrieves the current PCIe link generation
For Tesla™ and Quadro ® products from the Fermi and Kepler families.

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

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

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

Retrieves the current PCIe link width
For Tesla™ and Quadro ® products from the Fermi and Kepler families.

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

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

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

Retrieves the default applications clock that GPU boots with or defaults to after nvmlDeviceResetApplicationsClocks call.
For Tesla™ products from the Kepler family.
Module Documentation

Parameters:

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

Returns:

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

See also:

nvmlDeviceGetApplicationsClock

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

Retrieves the detailed ECC error counts for the device.

**Deprecated**

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

For Tesla™ and Quadro ® products from the Fermi and Kepler families, requires `NVML_INFOROM_ECC` version 2.0 or higher to report aggregate location-based ECC counts. Requires `NVML_INFOROM_ECC` version 1.0 or higher to report all other ECC counts. Requires ECC Mode to be enabled.

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

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

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

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

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

Parameters:

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

Returns:

- NVML_SUCCESS if **eccCounts** has been populated

NVIDIA Management Library
6.11 Device Queries

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

See also:

nvmlDeviceClearEccErrorCounts()

6.11.2.11 nvmlReturn_t DECLDIR nvmlDeviceGetDisplayActive (nvmlDevice_t device, nvmlEnableState_t *isActive)

Retrieves the display active state for the device.

For Tesla™ and Quadro ® products from the Fermi and Kepler families.

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

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

See `nvmlEnableState_t` for details on allowed modes.

Parameters:

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

Returns:

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

6.11.2.12 nvmlReturn_t DECLDIR nvmlDeviceGetDisplayMode (nvmlDevice_t device, nvmlEnableState_t *display)

Retrieves the display mode for the device.

For Tesla™ and Quadro ® products from the Fermi and Kepler families.

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

See `nvmlEnableState_t` for details on allowed modes.

Parameters:

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

Returns:

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

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

Retrieves the current and pending driver model for the device.

For Tesla™ and Quadro® products from the Fermi and Kepler families. For windows only.

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

See **nvmlDriverModel_t** for details on available driver models.

**Parameters:**

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

**Returns:**

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

See also:

**nvmlDeviceSetDriverModel()**

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

Retrieves the current and pending ECC modes for the device.

For Tesla™ and Quadro® products from the Fermi and Kepler families. Requires **NVML_INFOROM_ECC** version 1.0 or higher.

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

See **nvmlEnableState_t** for details on allowed modes.
6.11 Device Queries

Parameters:

device  The identifier of the target device

current Reference in which to return the current ECC mode

pending Reference in which to return the pending ECC mode

Returns:

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

See also:

nvmlDeviceSetEccMode()

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

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

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

For all discrete products with dedicated fans.

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

Parameters:

device  The identifier of the target device

speed Reference in which to return the fan speed percentage

Returns:

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

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

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

For GK110 M-class and X-class Tesla ™ products from the Kepler family. Not supported on Quadro ® and Tesla ™ C-class products.
6.11.2.17  

Parameters:

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

Returns:

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

See also:

- nvmlGpuOperationMode_t
- nvmlDeviceSetGpuOperationMode

6.11.2.17  

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

For all products.

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

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

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

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

- The target GPU is an SLI slave

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

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

Parameters:

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

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

See also:

- `nvmlDeviceGetIndex`
- `nvmlDeviceGetCount`

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

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

For all products.

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

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

- The target GPU is an SLI slave

Note:

NVML 4.304 and older version of `nvmlDeviceGetHandleByPciBusId"_v1"` returns **NVML_ERROR_NOTFOUND** instead of **NVML_ERROR_NO_PERMISSION**.

Parameters:

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

Returns:

- **NVML_SUCCESS** if `device` has been set
- **NVML_ERROR_UNINITIALIZED** if the library has not been successfully initialized
- **NVML_ERROR_INVALID_ARGUMENT** if `pciBusId` is invalid or `device` is NULL
- **NVML_ERROR_NOTFOUND** if `pciBusId` does not match a valid device on the system
- **NVML_ERROR_INSUFFICIENT_POWER** if the attached device has improperly attached external power cables
- **NVML_ERROR_NO_PERMISSION** if the user doesn’t have permission to talk to this device
- **NVML_ERROR_IRQ_ISSUE** if NVIDIA kernel detected an interrupt issue with the attached GPUs
- **NVML_ERROR_GPU_IS_LOST** if the target GPU has fallen off the bus or is otherwise inaccessible
- **NVML_ERROR_UNKNOWN** on any unexpected error
### 6.11.2.19 `nvmlReturn_t DECLDIR nvmlDeviceGetHandleBySerial (const char * serial, nvmlDevice_t * device)`

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

For all products.

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

**Deprecated**

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

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

**Parameters:**

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

**Returns:**

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

**See also:**

- `nvmlDeviceGetSerial`
- `nvmlDeviceGetHandleByUUID`

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

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

For all products.

**Parameters:**

- `uuid` The UUID of the target GPU
- `device` Reference in which to return the device handle
Starting from NVML 5, this API causes NVML to initialize the target GPU NVML may initialize additional GPUs as it searches for the target GPU.

Returns:

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

See also:

- `nvmlDeviceGetUUID`

6.11.2.21  
`nvmlReturn_t DECLDIR nvmlDeviceGetIndex (nvmlDevice_t device, unsigned int * index)`

Retrieves the NVML index of this device.

For all products.

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

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

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

Parameters:

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

Returns:

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

See also:

- `nvmlDeviceGetHandleByIndex()`
- `nvmlDeviceGetCount()`
6.11.2.22 nvmlReturn_t DECLDIR nvmlDeviceGetInforomConfigurationChecksum (nvmlDevice_t device, unsigned int *checksum)

Retrieves the checksum of the configuration stored in the device’s infoROM. For Tesla™ and Quadro ® products from the Fermi and Kepler families. Can be used to make sure that two GPUs have the exact same configuration. Current checksum takes into account configuration stored in PWR and ECC infoROM objects. Checksum can change between driver releases or when user changes configuration (e.g. disable/enable ECC)

Parameters:

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

Returns:

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

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

Retrieves the global infoROM image version For Tesla™ and Quadro ® products from the Kepler family. Image version just like VBIOS version uniquely describes the exact version of the infoROM flashed on the board in contrast to infoROM object version which is only an indicator of supported features. Version string will not exceed 16 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE.

Parameters:

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

Returns:

• NVML_SUCCESS if version has been set
• NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
• NVML_ERROR_INVALID_ARGUMENT if version is NULL
• NVML_ERROR_INSUFFICIENT_SIZE if length is too small
• NVML_ERROR_NOT_SUPPORTED if the device does not have an infoROM
6.11 Device Queries

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

See also:

nvmlDeviceGetInforomVersion

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

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

For Tesla™ and Quadro ® products from the Fermi and Kepler families.

Fermi and higher parts have non-volatile on-board memory for persisting device info, such as aggregate ECC counts.

The version of the data structures in this memory may change from time to time. It will not exceed 16 characters in

length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE.

See nvmlInforomObject_t for details on the available infoROM objects.

Parameters:

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

Returns:

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

See also:

nvmlDeviceGetInforomImageVersion

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

Retrieves the maximum clock speeds for the device.

For Tesla™ and Quadro ® products from the Fermi and Kepler families.

See nvmlClockType_t for details on available clock information.
Note:

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

Parameters:

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

Returns:

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

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

Retrieves the maximum PCIe link generation possible with this device and system
I.E. for a generation 2 PCIe device attached to a generation 1 PCIe bus the max link generation this function will report is generation 1.
For Tesla™ and Quadro ® products from the Fermi and Kepler families.

Parameters:

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

Returns:

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

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

Retrieves the maximum PCIe link width possible with this device and system
I.E. for a device with a 16x PCIe bus width attached to a 8x PCIe system bus this function will report a max link width of 8.
For Tesla™ and Quadro ® products from the Fermi and Kepler families.
Parameters:

device  The identifier of the target device

maxLinkWidth  Reference in which to return the max PCIe link generation

Returns:

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

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

Retrieves the requested memory error counter for the device.

For Tesla ™and Quadro ®products from the Fermi family. Requires NVML_INFOROM_ECC version 2.0 or higher to report aggregate location-based memory error counts. Requires NVML_INFOROM_ECC version 1.0 or higher to report all other memory error counts.

For all Tesla ™and Quadro ®products from the Kepler family.

Requires ECC Mode to be enabled.

See nvmlMemoryErrorType_t for a description of available memory error types.

See nvmlEccCounterType_t for a description of available counter types.

See nvmlMemoryLocation_t for a description of available counter locations.

Parameters:

device  The identifier of the target device

errorType  Flag that specifies the type of error.

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

locationType  Specifies the location of the counter.

count  Reference in which to return the ECC counter

Returns:

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

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

For all products.

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

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

See nvmlMemory_t for details on available memory info.

Parameters:

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

Returns:

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

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

Retrieves the name of this device.

For all products.

The name is an alphanumeric string that denotes a particular product, e.g. Tesla ™C2070. It will not exceed 64 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_NAME_BUFFER_SIZE.

Parameters:

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

Returns:

   • NVML_SUCCESS if name has been set
   • NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
   • NVML_ERROR_INVALID_ARGUMENT if device is invalid, or name is NULL
   • NVML_ERROR_INSUFFICIENT_SIZE if length is too small
   • NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
   • NVML_ERROR_UNKNOWN on any unexpected error
6.11.2.31 `nvmlReturn_t DECLDIR nvmlDeviceGetPciInfo (nvmlDevice_t device, nvmlPciInfo_t *pci)`

Retrieves the PCI attributes of this device.

For all products.

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

**Parameters:**

`device` The identifier of the target device

`pci` Reference in which to return the PCI info

**Returns:**

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

6.11.2.32 `nvmlReturn_t DECLDIR nvmlDeviceGetPerformanceState (nvmlDevice_t device, nvmlPstates_t *pState)`

Retrieves the current performance state for the device.

For Tesla™ and Quadro ® products from the Fermi and Kepler families.

See `nvmlPstates_t` for details on allowed performance states.

**Parameters:**

`device` The identifier of the target device

`pState` Reference in which to return the performance state reading

**Returns:**

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

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

Retrieves the persistence mode associated with this device.

For all CUDA-capable products. For Linux only.

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

See `nvmlEnableState_t` for details on allowed modes.
Parameters:

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

Returns:

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

See also:

- `nvmlDeviceSetPersistenceMode()`

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

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

For Tesla™ and Quadro ® products from the Kepler family.

Parameters:

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

Returns:

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

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

Retrieves the power management limit associated with this device.

For "GF11x" Tesla™ and Quadro ® products from the Fermi family.

- Requires **NVML_INFOROM_POWER** version 3.0 or higher.

For Tesla™ and Quadro ® products from the Kepler family.
6.11 Device Queries

• Does not require NVML_INFOROM_POWER object.

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

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

Parameters:

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

Returns:

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

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

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

For Tesla ™and Quadro ®products from the Kepler family.

Parameters:

device The identifier of the target device
minLimit Reference in which to return the minimum power management limit in milliwatts
maxLimit Reference in which to return the maximum power management limit in milliwatts

Returns:

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

See also:

nvmlDeviceSetPowerManagementLimit
6.11.2.37  nvmlReturn_t DECLDIR nvmlDeviceGetPowerManagementMode (nvmlDevice_t device,
        nvmlEnableState_t * mode)

Retrieves the power management mode associated with this device.
For "GF11x" Tesla ™ and Quadro ® products from the Fermi family.
  • Requires NVML_INFOROM_POWER version 3.0 or higher.
For Tesla ™ and Quadro ® products from the Kepler family.
  • Does not require NVML_INFOROM_POWER object.

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

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

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

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

Deprecated: Use nvmlDeviceGetPerformanceState. This function exposes an incorrect generalization.
Retrieve the current performance state for the device.
For Tesla ™ and Quadro ® products from the Fermi and Kepler families.
See nvmlPstates_t for details on allowed performance states.

Parameters:
  
  device  The identifier of the target device
  
  pState  Reference in which to return the performance state reading

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

NVIDIA Management Library
6.11.2.39  `nvmlReturn_t DECLDIR nvmlDeviceGetPowerUsage (nvmlDevice_t device, unsigned int * power)`

Retrieves power usage for this GPU in milliwatts and its associated circuitry (e.g. memory)
For "GF11x" Tesla™ and Quadro ® products from the Fermi family.

- Requires `NVML_INFOROM_POWER` version 3.0 or higher.

For Tesla™ and Quadro ® products from the Kepler family.

- Does not require `NVML_INFOROM_POWER` object.

On Fermi and Kepler GPUs the reading is accurate to within +/- 5% of current power draw.
It is only available if power management mode is supported. See `nvmlDeviceGetPowerManagementMode`.

**Parameters:**

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

**Returns:**

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

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

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

For Tesla™K20 products

**Parameters:**

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

**Returns:**

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

6.11.2.41 nvmlReturn_t DECLDIR nvmlDeviceGetRetiredPagesPendingStatus (nvmlDevice_t device, nvmlEnableState_t *isPending)

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

For Tesla™K20 products

Parameters:

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

Returns:

• NVML_SUCCESS if isPending was populated
• NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
• NVML_ERROR_INVALID_ARGUMENT if device is invalid or isPending is NULL
• NVML_ERROR_NOT_SUPPORTED if the device doesn’t support this feature
• NVML_ERROR_GPU_IS_LOST if the target GPU has fallen off the bus or is otherwise inaccessible
• NVML_ERROR_UNKNOWN on any unexpected error

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

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

For Tesla™and Quadro® products from the Fermi and Kepler families.

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

Parameters:

   device The identifier of the target device
   serial Reference in which to return the board/module serial number
   length The maximum allowed length of the string returned in serial

Returns:

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

6.11.2.43 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 devices

Parameters:

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

Returns:

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

See also:

NvmlClocksThrottleReasons
nvmlDeviceGetCurrentClocksThrottleReasons

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

Retrieves the list of possible graphics clocks that can be used as an argument for nvmlDeviceSetApplicationsClocks.
For Tesla ™ products and Quadro ® products from the Kepler family.

Parameters:

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

Returns:

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

See also:
  nvmlDeviceSetApplicationsClocks
  nvmlDeviceGetSupportedMemoryClocks

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

Retrieves the list of possible memory clocks that can be used as an argument for nvmlDeviceSetApplicationsClocks. For Tesla™ products from the Kepler family.

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

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

See also:
  nvmlDeviceSetApplicationsClocks
  nvmlDeviceGetSupportedGraphicsClocks

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

Retrieves the current temperature readings for the device, in degrees C. For all discrete and S-class products.

See nvmlTemperatureSensors_t for details on available temperature sensors.

Parameters:
  device  The identifier of the target device
sensorType  Flag that indicates which sensor reading to retrieve

temp  Reference in which to return the temperature reading

Returns:

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

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

Retrieves the total ECC error counts for the device.

For Tesla ™ and Quadro ® products from the Fermi and Kepler families. Requires NVML_INFOROM_ECC version 1.0 or higher. Requires ECC Mode to be enabled.

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

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

Parameters:

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

Returns:

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

See also:

nvmlDeviceClearEccErrorCounts()
6.11.2.48  nvmlReturn_t DECLDIR nvmlDeviceGetUtilizationRates (nvmlDevice_t device, nvmlUtilization_t *utilization)

Retrieves the current utilization rates for the device’s major subsystems. For Tesla™ and Quadro ® products from the Fermi and Kepler families. See nvmlUtilization_t for details on available utilization rates.

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

**Parameters:**
- **device**  The identifier of the target device
- **utilization**  Reference in which to return the utilization information

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

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

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

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

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

**Returns:**
- **NVML_SUCCESS** if uuid has been set
- **NVML_ERROR_UNINITIALIZED** if the library has not been successfully initialized
- **NVML_ERROR_INVALID_ARGUMENT** if device is invalid, or uuid is NULL
- **NVML_ERROR_INSUFFICIENT_SIZE** if length is too small
- **NVML_ERROR_NOT_SUPPORTED** if the device does not support this feature
- **NVML_ERROR_GPU_IS_LOST** if the target GPU has fallen off the bus or is otherwise inaccessible
- **NVML_ERROR_UNKNOWN** on any unexpected error
6.11.2.50 nvmlReturn_t DECLDIR nvmlDeviceGetVbiosVersion (nvmlDevice_t device, char ∗ version, unsigned int length)

Get VBIOS version of the device.
For all products.
The VBIOS version may change from time to time. It will not exceed 32 characters in length (including the NULL terminator). See nvmlConstants::NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE.

Parameters:

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

Returns:

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

6.11.2.51 nvmlReturn_t DECLDIR nvmlDeviceOnSameBoard (nvmlDevice_t device1, nvmlDevice_t device2, int ∗ onSameBoard)

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

Parameters:

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

Returns:

- **NVML_SUCCESS** if **onSameBoard** has been set
- **NVML_ERROR_UNINITIALIZED** if the library has not been successfully initialized
- **NVML_ERROR_INVALID_ARGUMENT** if **device1** or **device2** are invalid or **onSameBoard** is NULL
- **NVML_ERROR_NOT_SUPPORTED** if this check is not supported by the device
- **NVML_ERROR_GPU_IS_LOST** if the either GPU has fallen off the bus or is otherwise inaccessible
- **NVML_ERROR_UNKNOWN** on any unexpected error
6.11.2.52 nvmlReturn_t DECLDIR nvmlDeviceResetApplicationsClocks (nvmlDevice_t device)

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

See also:
   nvmlDeviceGetApplicationsClock
   nvmlDeviceSetApplicationsClocks

For Tesla™ products from the Kepler family.

Parameters:
   device The identifier of the target device

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

6.11.2.53 nvmlReturn_t DECLDIR nvmlDeviceValidateInfoROM (nvmlDevice_t device)

Reads the infoROM from the flash and verifies the checksums.
For Tesla™ and Quadro® products from the Fermi and Kepler families.

Parameters:
   device The identifier of the target device

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

Functions

- `nvmlReturn_t DECLDIR nvmlUnitSetLedState (nvmlUnit_t unit, nvmlLedColor_t color)`

6.12.1 Detailed Description

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

6.12.2 Function Documentation

6.12.2.1 `nvmlReturn_t DECLDIR nvmlUnitSetLedState (nvmlUnit_t unit, nvmlLedColor_t color)`

Set the LED state for the unit. The LED can be either green (0) or amber (1).
For S-class products. Requires root/admin permissions.
This operation takes effect immediately.
Current S-Class products don't provide unique LEDs for each unit. As such, both front and back LEDs will be toggled in unison regardless of which unit is specified with this command.
See `nvmlLedColor_t` for available colors.

Parameters:

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

Returns:

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

See also:

`nvmlUnitGetLedState()`
6.13 Device Commands

Functions

- `nvmlReturn_t DECLDIR nvmlDeviceSetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t mode)`
- `nvmlReturn_t DECLDIR nvmlDeviceSetComputeMode (nvmlDevice_t device, nvmlComputeMode_t mode)`
- `nvmlReturn_t DECLDIR nvmlDeviceSetEccMode (nvmlDevice_t device, nvmlEnableState_t ecc)`
- `nvmlReturn_t DECLDIR nvmlDeviceClearEccErrorCounts (nvmlDevice_t device, nvmlEccCounterType_t counterType)`
- `nvmlReturn_t DECLDIR nvmlDeviceSetDriverModel (nvmlDevice_t device, nvmlDriverModel_t driverModel, unsigned int flags)`
- `nvmlReturn_t DECLDIR nvmlDeviceSetApplicationsClocks (nvmlDevice_t device, unsigned int memClockMHz, unsigned int graphicsClockMHz)`
- `nvmlReturn_t DECLDIR nvmlDeviceSetPowerManagementLimit (nvmlDevice_t device, unsigned int limit)`
- `nvmlReturn_t DECLDIR nvmlDeviceSetGpuOperationMode (nvmlDevice_t device, nvmlGpuOperationMode_t mode)`

6.13.1 Detailed Description

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

6.13.2 Function Documentation

6.13.2.1 `nvmlReturn_t DECLDIR nvmlDeviceClearEccErrorCounts (nvmlDevice_t device, nvmlEccCounterType_t counterType)`

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

For Tesla ™and Quadro ®products from the Fermi and Kepler families. Requires NVML_INFOROM_ECC version 2.0 or higher to clear aggregate location-based ECC counts. Requires NVML_INFOROM_ECC version 1.0 or higher to clear all other ECC counts. Requires root/admin permissions. Requires ECC Mode to be enabled.

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

This operation takes effect immediately.

See `nvmlMemoryErrorType_t` for details on available counter types.

Parameters:

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

Returns:

- `NVML_SUCCESS` if the error counts were cleared
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if `device` is invalid or `counterType` is invalid
- `NVML_ERROR_NOT_SUPPORTED` if the device does not support this feature
- `NVML_ERROR_NO_PERMISSION` if the user doesn’t have permission to perform this operation
- `NVML_ERROR_GPU_IS_LOST` if the target GPU has fallen off the bus or is otherwise inaccessible
6.13 Device Commands

- NVML_ERROR_UNKNOWN on any unexpected error

See also:
- nvmlDeviceGetDetailedEccErrors()
- nvmlDeviceGetTotalEccErrors()

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

Set clocks that applications will lock to.
Sets the clocks that compute and graphics applications will be running at. e.g. CUDA driver requests these clocks during context creation which means this property defines clocks at which CUDA applications will be running unless some overspec event occurs (e.g. over power, over thermal or external HW brake).
Can be used as a setting to request constant performance.
For Tesla™ products from the Kepler family. Requires root/admin permissions.
See nvmlDeviceGetSupportedMemoryClocks and nvmlDeviceGetSupportedGraphicsClocks for details on how to list available clocks combinations.
After system reboot or driver reload applications clocks go back to their default value. See nvmlDeviceResetApplicationsClocks.

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

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

6.13.2.3 nvmlReturn_t DECLDIR nvmlDeviceSetComputeMode (nvmlDevice_t device, nvmlComputeMode_t mode)

Set the compute mode for the device.
For all CUDA-capable products. Requires root/admin permissions.
The compute mode determines whether a GPU can be used for compute operations and whether it can be shared across contexts.
This operation takes effect immediately. Under Linux it is not persistent across reboots and always resets to "Default". Under windows it is persistent.

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

See nvmlComputeMode_t for details on available compute modes.

Parameters:

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

Returns:

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

See also:

nvmlDeviceGetComputeMode()

6.13.2.4  **nvmlReturn_t DECLDIR nvmlDeviceSetDriverModel (nvmlDevice_t device, nvmlDriverModel_t driverModel, unsigned int flags)**

Set the driver model for the device.

For Tesla TM and Quadro ® products from the Fermi and Kepler families. For windows only. Requires root/admin permissions.

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

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

This operation takes effect after the next reboot.

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

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

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

Parameters:

- **device**  The identifier of the target device
- **driverModel**  The target driver model
- **flags**  Flags that change the default behavior
6.13 Device Commands

Returns:

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

See also:

nvmlDeviceGetDriverModel()

6.13.2.5 nvmlReturn_t DECLDIR nvmlDeviceSetEccMode (nvmlDevice_t device, nvmlEnableState_t ecc)

Set the ECC mode for the device.

For Tesla™ and Quadro® products from the Fermi and Kepler families. Requires NVML_INFOROM_ECC version 1.0 or higher. Requires root/admin permissions.

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

This operation takes effect after the next reboot.

See nvmlEnableState_t for details on available modes.

Parameters:

device The identifier of the target device
decc The target ECC mode

Returns:

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

See also:

nvmlDeviceGetEccMode()
6.13.2.6 nvmlReturn_t DECLDIR nvmlDeviceSetGpuOperationMode (nvmlDevice_t device, nvmlGpuOperationMode_t mode)

Sets new GOM. See nvmlGpuOperationMode_t for details.

For GK110 M-class and X-class Tesla™ products from the Kepler family. Not supported on Quadro® and Tesla™ C-class products. Requires root/admin permissions.

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

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

Parameters:

- **device** The identifier of the target device
- **mode** Target GOM

Returns:

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

See also:

- nvmlGpuOperationMode_t
- nvmlDeviceGetGpuOperationMode

6.13.2.7 nvmlReturn_t DECLDIR nvmlDeviceSetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t mode)

Set the persistence mode for the device.

For all CUDA-capable products. For Linux only. Requires root/admin permissions.

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

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

See nvmlEnableState_t for available modes.

Parameters:

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

Returns:

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

See also:
`nvmlDeviceGetPersistenceMode()`

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

Set new power limit of this device.
For Tesla™and Quadro® products from the Kepler family. Requires root/admin permissions.
See `nvmlDeviceGetPowerManagementLimitConstraints` to check the allowed ranges of values.

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

Parameters:

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

Returns:

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

See also:
`nvmlDeviceGetPowerManagementLimitConstraints`
`nvmlDeviceGetPowerManagementDefaultLimit`
6.14 Event Handling Methods

Data Structures

- struct nvmlEventData_t

Modules

- Event Types

Typedefs

- typedef struct nvmlEventSet_st *nvmlEventSet_t

Functions

- nvmlReturn_t DECLDIR nvmlEventSetCreate (nvmlEventSet_t *set)
- nvmlReturn_t DECLDIR nvmlDeviceRegisterEvents (nvmlDevice_t device, unsigned long long eventTypes, nvmlEventSet_t set)
- nvmlReturn_t DECLDIR nvmlDeviceGetSupportedContentTypes (nvmlDevice_t device, unsigned long long *eventTypes)
- nvmlReturn_t DECLDIR nvmlEventSetWait (nvmlEventSet_t set, nvmlEventData_t *data, unsigned int timeoutms)
- nvmlReturn_t DECLDIR nvmlEventSetFree (nvmlEventSet_t set)

6.14.1 Detailed Description

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

6.14.2 Typedef Documentation

6.14.2.1 typedef struct nvmlEventSet_st * nvmlEventSet_t

Handle to an event set

6.14.3 Function Documentation

6.14.3.1 nvmlReturn_t DECLDIR nvmlDeviceGetSupportedContentTypes (nvmlDevice_t device, unsigned long long *eventTypes)

Returns information about events supported on device

For all products.

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

Parameters:

device The identifier of the target device
**6.14 Event Handling Methods**

**eventTypes**  Reference in which to return bitmask of supported events

**Returns:**

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

See also:

- Event Types
- `nvmlDeviceRegisterEvents`

### 6.14.3.2 `nvmlReturn_t DECLDIR nvmlDeviceRegisterEvents (nvmlDevice_t device, unsigned long long eventTypes, nvmlEventSet_t set)`

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

For Tesla™ and Quadro ® products from the Fermi and Kepler families. Ecc events are available only on ECC enabled devices (see `nvmlDeviceGetTotalEccErrors`) Power capping events are available only on Power Management enabled devices (see `nvmlDeviceGetPowerManagementMode`)

For Linux only.

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

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

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

**Parameters:**

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

**Returns:**

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

See also:

- Event Types
- `nvmlDeviceGetSupportedEventTypes`
- `nvmlEventSetWait`
- `nvmlEventSetFree`
6.14.3.3 nvmlReturn_t DECLDIR nvmlEventSetCreate (nvmlEventSet_t * set)

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

Parameters:

set Reference in which to return the event handle

Returns:

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

See also:

nvmlEventSetFree

6.14.3.4 nvmlReturn_t DECLDIR nvmlEventSetFree (nvmlEventSet_t set)

Releases events in the set

For Tesla™ and Quadro ® products from the Fermi and Kepler families.

Parameters:

set Reference to events to be released

Returns:

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

See also:

nvmlDeviceRegisterEvents

6.14.3.5 nvmlReturn_t DECLDIR nvmlEventSetWait (nvmlEventSet_t set, nvmlEventData_t * data, unsigned int timeoutms)

Waits on events and delivers events

For Tesla™ and Quadro ® products from the Fermi and Kepler families.

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

Parameters:

set Reference to set of events to wait on

data Reference in which to return event data
6.14 Event Handling Methods

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

**Returns:**

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

**See also:**

- Event Types
- `nvmlDeviceRegisterEvents`
6.15 NvmlClocksThrottleReasons

Defines

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

6.15.1 Define Documentation

6.15.1.1 #define nvmlClocksThrottleReasonAll

Value:

\[
(nvmlClocksThrottleReasonNone \  \\
| nvmlClocksThrottleReasonGpuIdle \  \\
| nvmlClocksThrottleReasonApplicationsClocksSetting \  \\
| nvmlClocksThrottleReasonSwPowerCap \  \\
| nvmlClocksThrottleReasonHwSlowdown \  \\
| nvmlClocksThrottleReasonUnknown)
\]

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

6.15.1.2 #define nvmlClocksThrottleReasonApplicationsClocksSetting 0x0000000000000002LL

GPU clocks are limited by current setting of applications clocks

See also:

- `nvmlDeviceSetApplicationsClocks`
- `nvmlDeviceGetApplicationsClock`

6.15.1.3 #define nvmlClocksThrottleReasonGpuIdle 0x0000000000000001LL

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

Note:

This limiter may be removed in a later release

6.15.1.4 #define nvmlClocksThrottleReasonHwSlowdown 0x0000000000000008LL

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

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

See also:
  nvmlDeviceGetTemperature
  nvmlDeviceGetPowerUsage

6.15.1.5  #define nvmlClocksThrottleReasonNone 0x0000000000000000LL

Bit mask representing no clocks throttling
Clocks are as high as possible.

6.15.1.6  #define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL

SW Power Scaling algorithm is reducing the clocks below requested clocks

See also:
  nvmlDeviceGetPowerUsage
  nvmlDeviceGetPowerManagementLimit
  nvmlDeviceSetPowerManagementLimit

6.15.1.7  #define nvmlClocksThrottleReasonUnknown 0x8000000000000000LL

Some other unspecified factor is reducing the clocks

6.15.1.8  #define nvmlClocksThrottleReasonUserDefinedClocks nvmlClocksThrottleReasonApplication-ClocksSetting

Deprecated

Renamed to nvmlClocksThrottleReasonApplicationsClocksSetting as the name describes the situation more accurately.
Chapter 7

Data Structure Documentation

7.1 nvmlAccountingStats_t Struct Reference

#include <nvml.h>

Data Fields

- unsigned int gpuUtilization
  
  Percent of time over the process’s lifetime during which one or more kernels was executing on the GPU. Utilization stats just like returned by nvmlDeviceGetUtilizationRates but for the life time of a process (not just the last sample period). Set to NVML_VALUE_NOT_AVAILABLE if nvmlDeviceGetUtilizationRates is not supported.

- unsigned int memoryUtilization
  
  Percent of time over the process’s lifetime during which global (device) memory was being read or written. Set to NVML_VALUE_NOT_AVAILABLE if nvmlDeviceGetUtilizationRates is not supported.

- unsigned long long maxMemoryUsage
  
  Maximum total memory in bytes that was ever allocated by the process. Set to NVML_VALUE_NOT_AVAILABLE if nvmlProcessInfo_t->usedGpuMemory is not supported.

- unsigned long long time
  
  Amount of time in ms during which the compute context was active.

7.1.1 Detailed Description

Describes accounting statistics of a process.
# Data Structure Documentation

## 7.2 nvmlEccErrorCounts_t Struct Reference

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

### Data Fields

- `unsigned long long l1Cache`
  
  *L1 cache errors.*

- `unsigned long long l2Cache`
  
  *L2 cache errors.*

- `unsigned long long deviceMemory`
  
  *Device memory errors.*

- `unsigned long long registerFile`
  
  *Register file errors.*

### 7.2.1 Detailed Description

Detailed ECC error counts for a device.

**Deprecated**

Different GPU families can have different memory error counters. See `nvmlDeviceGetMemoryErrorCounter`
#include <nvml.h>

## Data Fields

- `nvmlDevice_t device`
  
  *Specific device where the event occurred.*

- `unsigned long long eventType`
  
  *Information about what specific event occurred.*

## Detailed Description

Information about occurred event
7.4 nvmlHwbcEntry_t Struct Reference

#include <nvml.h>

7.4.1 Detailed Description

Description of HWBC entry
Data Fields

- **char cause [256]**
  
  *If amber, a text description of the cause.*

- **nvmlLedColor_t color**
  
  *GREEN or AMBER.*

7.5.1 Detailed Description

LED states for an S-class unit.
Data Structure Documentation

7.6 nvmlMemory_t Struct Reference

#include <nvml.h>

Data Fields

- unsigned long long total
  Total installed FB memory (in bytes).

- unsigned long long free
  Unallocated FB memory (in bytes).

- unsigned long long used
  Allocated FB memory (in bytes). Note that the driver/GPU always sets aside a small amount of memory for bookkeeping.

7.6.1 Detailed Description

Memory allocation information for a device.
7.7  nvmlPciInfo_t Struct Reference

#include <nvml.h>

Data Fields

- char busId [NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE]
  The tuple domain:bus:device:function PCI identifier (& NULL terminator).

- unsigned int domain
  The PCI domain on which the device's bus resides, 0 to 0xffff.

- unsigned int bus
  The bus on which the device resides, 0 to 0xff.

- unsigned int device
  The device's id on the bus, 0 to 31.

- unsigned int pciDeviceId
  The combined 16-bit device id and 16-bit vendor id.

- unsigned int pciSubSystemId
  The 32-bit Sub System Device ID.

7.7.1 Detailed Description

PCI information about a GPU device.
7.8  nvmlProcessInfo_t Struct Reference

#include <nvml.h>

Data Fields

- unsigned int pid
  
  Process ID.

- unsigned long long usedGpuMemory

  Amount of used GPU memory in bytes. Under WDDM, NVML_VALUE_NOT_AVAILABLE is always reported because
  Windows KMD manages all the memory and not the NVIDIA driver.

7.8.1  Detailed Description

Information about running compute processes on the GPU
#include <nvml.h>

Data Fields

- char state [256]
  
  *The power supply state.*

- unsigned int current
  
  *PSU current (A).*

- unsigned int voltage
  
  *PSU voltage (V).*

- unsigned int power
  
  *PSU power draw (W).*

## 7.9.1 Detailed Description

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

- High voltage
- Fan failure
- Heatsink temperature
- Current limit
- Voltage below UV alarm threshold
- Low-voltage
- SI2C remote off command
- MOD_DISABLE input
- Short pin transition
7.10  `nvmlUnitFanInfo_t` Struct Reference

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

**Data Fields**

- `unsigned int speed`
  
  *Fan speed (RPM)*.

- `nvmlFanState_t state`
  
  *Flag that indicates whether fan is working properly.*

### 7.10.1 Detailed Description

Fan speed reading for a single fan in an S-class unit.
7.11 nvmlUnitFanSpeeds_t Struct Reference

#include <nvml.h>

Data Fields

- **nvmlUnitFanInfo_t fans [24]**
  
  Fan speed data for each fan.

- **unsigned int count**
  
  Number of fans in unit.

7.11.1 Detailed Description

Fan speed readings for an entire S-class unit.
7.12 nvmlUnitInfo_t Struct Reference

#include <nvml.h>

Data Fields

- char name [96]  
  Product name.

- char id [96]  
  Product identifier.

- char serial [96]  
  Product serial number.

- char firmwareVersion [96]  
  Firmware version.

7.12.1 Detailed Description

Static S-class unit info.
#include <nvml.h>

Data Fields

- unsigned int `gpu`
  
  Percent of time over the past sample period during which one or more kernels was executing on the GPU.

- unsigned int `memory`
  
  Percent of time over the past sample period during which global (device) memory was being read or written.

7.13.1 Detailed Description

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

Accounting Statistics, 24
Constants, 31
Device Commands, 70
Device Enums, 14
Device Queries, 38
Device Structs, 13
Error reporting, 30
Event Handling Methods, 76
Event Types, 22
Initialization and Cleanup, 28
NVML_AGGREGATE_ECC
  nvmlDeviceEnumvs, 17
NVML_CLOCK_GRAPHICS
  nvmlDeviceEnumvs, 16
NVML_CLOCK_MEM
  nvmlDeviceEnumvs, 16
NVML_CLOCK_SM
  nvmlDeviceEnumvs, 16
NVML_COMPUTEMODE_DEFAULT
  nvmlDeviceEnumvs, 17
NVML_COMPUTEMODE_EXCLUSIVE_PROCESS
  nvmlDeviceEnumvs, 17
NVML_COMPUTEMODE_EXCLUSIVE_THREAD
  nvmlDeviceEnumvs, 17
NVML_COMPUTEMODE_PROHIBITED
  nvmlDeviceEnumvs, 17
NVML_DRIVER_WDDM
  nvmlDeviceEnumvs, 17
NVML_DRIVER_WDM
  nvmlDeviceEnumvs, 17
NVML_ECC_COUNTER_TYPE_COUNT
  nvmlDeviceEnumvs, 17
NVML_ERROR_ALREADY_INITIALIZED
  nvmlDeviceEnumvs, 19
NVML_ERROR_CORRUPTED_INFOROM
  nvmlDeviceEnumvs, 20
NVML_ERROR_DRIVER_NOT_LOADED
  nvmlDeviceEnumvs, 20
NVML_ERROR_FUNCTION_NOT_FOUND
  nvmlDeviceEnumvs, 20
NVML_ERROR_GPU_IS_LOST
  nvmlDeviceEnumvs, 20
NVML_ERROR_INSUFFICIENT_POWER
  nvmlDeviceEnumvs, 20
NVML_ERROR_INSUFFICIENT_SIZE
  nvmlDeviceEnumvs, 20
NVML_ERROR_INVALID_ARGUMENT
  nvmlDeviceEnumvs, 19
NVML_ERROR_IRQ_ISSUE
  nvmlDeviceEnumvs, 20
NVML_ERROR_LIBRARY_NOT_FOUND
  nvmlDeviceEnumvs, 20
NVML_ERROR_NO_PERMISSION
  nvmlDeviceEnumvs, 19
NVML_ERROR_NOT_FOUND
  nvmlDeviceEnumvs, 20
NVML_ERROR_NOT_SUPPORTED
  nvmlDeviceEnumvs, 19
NVML_ERROR_TIMEOUT
  nvmlDeviceEnumvs, 20
NVML_ERROR_UNINITIALIZED
  nvmlDeviceEnumvs, 19
NVML_ERROR_UNKNOWN
  nvmlDeviceEnumvs, 20
NVML_FAN_FAILED
  nvmlUnitStructs, 21
NVML_FAN_NORMAL
  nvmlUnitStructs, 21
NVML_FEATURE_DISABLED
  nvmlDeviceEnumvs, 17
NVML_FEATURE_ENABLED
  nvmlDeviceEnumvs, 17
NVML_GOM_ALL_ON
  nvmlDeviceEnumvs, 18
NVML_GOM_COMPUTE
  nvmlDeviceEnumvs, 18
NVML_GOM_LOW_DP
  nvmlDeviceEnumvs, 18
NVML_INFOROM_COUNT
  nvmlDeviceEnumvs, 18
NVML_INFOROM_ECC
  nvmlDeviceEnumvs, 18
NVML_INFOROM_OEM
  nvmlDeviceEnumvs, 18
NVML_INFOROM_POWER
  nvmlDeviceEnumvs, 18
nvmlClocksThrottleReasons, 80
nvmlClocksThrottleReasonNone
nvmlClocksThrottleReasons, 81
NvmlClocksThrottleReasons, 80
nvmlClocksThrottleReasons
nvmlClocksThrottleReasonAll, 80
nvmlClocksThrottleReasonApplicationsClocksSetting, 80
nvmlClocksThrottleReasonGpuIdle, 80
nvmlClocksThrottleReasonHwSlowdown, 80
nvmlClocksThrottleReasonNone, 81
nvmlClocksThrottleReasonSwPowerCap, 81
nvmlClocksThrottleReasonUnknown, 81
nvmlClocksThrottleReasonUserDefinedClocks, 81
nvmlClocksThrottleReasonSwPowerCap
nvmlClocksThrottleReasons, 81
nvmlClocksThrottleReasonUnknown
nvmlClocksThrottleReasons, 81
nvmlClocksThrottleReasonUserDefinedClocks
nvmlClocksThrottleReasons, 81
nvmlClockType_t
nvDeviceEnumvs, 16
nvmlComputeMode_t
nvDeviceEnumvs, 16
nvmlConstants
NVML_DEVICE_INFOROM_VERSION_-BUFFER_SIZE, 31
NVML_DEVICE_NAME_BUFFER_SIZE, 31
NVML_DEVICE_SERIAL_BUFFER_SIZE, 31
NVML_DEVICE_UUID_BUFFER_SIZE, 31
NVML DEVICE VBIOS VERSION BUFFER_- SIZE, 31
NVML_SYSTEM_DRIVER VERSION_-BUFFER_SIZE, 31
NVML_SYSTEM_NVML VERSION BUFFER_- SIZE, 31
nvmlDeviceClearAccountingPids
nvmlAccountingStats, 24
nvmlDeviceClearEccErrorCounts
nvmlDeviceCommands, 70
nvmlDeviceCommands
nvmlDeviceClearEccErrorCounts, 70
nvmlDeviceSetApplicationsClocks, 71
nvmlDeviceSetComputeMode, 71
nvmlDeviceSetDriverModel, 72
nvmlDeviceSetEccMode, 73
nvmlDeviceSetGpuOperationMode, 73
nvmlDeviceSetPersistenceMode, 74
nvmlDeviceSetPowerManagementLimit, 75
nvmlDeviceEnumvs
NVML_AGGREGATE_ECC, 17
NVML_CLOCK_GRAPHICS, 16
NVML CLOCK_MEM, 16
NVML CLOCK SM, 16
NVML COMPUTEMODE_DEFAULT, 17
NVML COMPUTEMODE EXCLUSIVE_-PROCESS, 17
NVML COMPUTEMODE EXCLUSIVE_-THREAD, 17
NVML COMPUTEMODE_PROHIBITED, 17
NVML DRIVER_WDDM, 17
NVML DRIVER_WDM, 17
NVML ECC_COUNTER_TYPE_COUNT, 17
NVML ERROR_ALREADY_INITIALIZED, 19
NVML_ERROR_CORRUPTED_INFOROM, 20
NVML ERROR DRIVER NOT LOADED, 20
NVML ERROR FUNCTION NOT FOUND, 20
NVML ERROR_GPU IS LOST, 20
NVML ERROR_INSUFFICIENT_POWER, 20
NVML ERROR INSUFFICIENT_SIZE, 20
NVML ERROR INVALID_ARGUMENT, 19
NVML ERROR IRQ ISSUE, 20
NVML ERROR_LIBRARY_NOT_FOUND, 20
NVML ERROR NO_PERMISSION, 19
NVML ERROR NOT FOUND, 20
NVML ERROR NOT SUPPORTED, 19
NVML ERROR_TIMEOUT, 20
NVML ERROR UNINITIALIZED, 19
NVML ERROR UNKNOWN, 20
NVML FEATURE DISABLED, 17
NVML FEATURE ENABLED, 17
NVML GOM ALL ON, 18
NVML GOM COMPUTE, 18
NVML GOM LOW DP, 18
NVML INFOROM COUNT, 18
NVML INFOROM ECC, 18
NVML INFOROM OEM, 18
NVML INFOROM POWER, 18
NVML MEMORY ERROR TYPE_-CORRECTED, 18
NVML MEMORY ERROR_TYPE_COUNT, 18
NVML MEMORY ERROR TYPE_-UNCORRECTED, 18
NVML MEMORY LOCATION_COUNT, 18
NVML MEMORY LOCATION DEVICE_-MEMORY, 18
NVML MEMORY LOCATION L1 CACHE, 18
NVML MEMORY LOCATION L2 CACHE, 18
NVML MEMORY LOCATION REGISTER_-FILE, 18
NVML MEMORY LOCATION_TEXTURE_-MEMORY, 18
NVML_PAGE RETIREMENT CAUSE_-DOUBLE_BIT ECC ERROR, 19
NVML PAGE RETIREMENT CAUSE_-MULTIPLE SINGLE_BIT ECC_ERRORS, 19
NVML_PSTATE_0, 19
NVIDIA Management Library
nvmlDeviceCommands, 73
nvmlDeviceSetGpuOperationMode
nvmlDeviceCommands, 73
nvmlDeviceSetPersistenceMode
nvmlDeviceCommands, 74
nvmlDeviceSetPowerManagementLimit
nvmlDeviceCommands, 75
nvmlDeviceStructs
NVML_DEVICE_PCI_BUS_ID_BUFFER_SIZE, 13
NVML_VALUE_NOT_AVAILABLE, 13
nvmlDeviceValidateInforom
nvmlDeviceQueries, 68
nvmlDriverModel_t
nvmlDeviceEnumvs, 17
nvmlEccBitType_t
nvmlDeviceEnumvs, 16
nvmlEccCounterType_t
nvmlDeviceEnumvs, 17
nvmlEccErrorCounts_t, 84
nvmlEnableState_t
nvmlDeviceEnumvs, 17
nvmlErrorReporting
nvmlErrorString, 30
nvmlErrorString
nvmlErrorReporting, 30
nvmlEventData_t, 85
nvmlEvents
nvmlDeviceGetSupportedEventTypes, 76
nvmlDeviceRegisterEvents, 77
nvmlEventSet_t, 76
nvmlEventSetCreate, 77
nvmlEventSetFree, 78
nvmlEventSetWait, 78
nvmlEventSet_t
nvmlEvents, 76
nvmlEventSetCreate
nvmlEvents, 77
nvmlEventSetFree
nvmlEvents, 78
nvmlEventSetWait
nvmlEvents, 78
nvmlEventType
nvmlEventTypeClock, 22
nvmlEventTypeDoubleBitEccError, 22
nvmlEventTypePState, 22
nvmlEventTypeSingleBitEccError, 23
nvmlEventTypeClock
nvmlEventType, 22
nvmlEventTypeDoubleBitEccError
nvmlEventType, 22
nvmlEventTypePState
nvmlEventType, 22
nvmlEventTypeSingleBitEccError
nvmlEventType, 23
nvmlFanState_t
nvmlUnitStructs, 21
nvmlGpuOperationMode_t
nvmlDeviceEnumvs, 17
nvmlHwbcEntry_t, 86
nvmlInforomObject_t
nvmlDeviceEnumvs, 18
nvmlInit
nvmlInitializationAndCleanup, 28
nvmlInitializationAndCleanup
nvmlInit, 28
nvmlShutdown, 28
nvmlLedColor_t
nvmlUnitStructs, 21
nvmlLedState_t, 87
nvmlMemory_t, 88
nvmlMemoryErrorType_t
nvmlDeviceEnumvs, 18
nvmlMemoryLocation_t
nvmlDeviceEnumvs, 18
nvmlPageRetirementCause_t
nvmlDeviceEnumvs, 18
nvmlPciInfo_t, 89
nvmlProcessInfo_t, 90
nvmlPstates_t
nvmlDeviceEnumvs, 19
nvmlPSUIfo_t, 91
nvmlReturn_t
nvmlDeviceEnumvs, 19
nvmlShutdown
nvmlInitializationAndCleanup, 28
nvmlSystemGetDriverVersion
nvmlSystemQueries, 32
nvmlSystemGetHicVersion
nvmlUnitQueries, 34
nvmlSystemGetNVMLVersion
nvmlSystemQueries, 32
nvmlSystemGetProcessName
nvmlSystemQueries, 33
nvmlSystemQueries
nvmlSystemGetDriverVersion, 32
nvmlSystemGetNVMLVersion, 32
nvmlSystemGetProcessName, 33
nvmlTemperatureSensors_t
nvmlDeviceEnumvs, 20
nvmlUnitCommands
nvmlUnitSetLedState, 69
nvmlUnitFanInfo_t, 92
nvmlUnitFanSpeeds_t, 93
nvmlUnitGetCount
nvmlUnitQueries, 34
nvmlUnitGetDevices
nvmlUnitQueries, 35
nvmlUnitGetFanSpeedInfo
nvmlUnitQueries, 35
nvmlUnitGetHandleByIndex
nvmlUnitQueries, 35
nvmlUnitGetLedState
nvmlUnitQueries, 36
nvmlUnitGetPsuInfo
nvmlUnitQueries, 36
nvmlUnitGetTemperature
nvmlUnitQueries, 37
nvmlUnitGetUnitInfo
nvmlUnitQueries, 37
nvmlUnitInfo_t, 94
nvmlUnitQueries
nvmlSystemGetHicVersion, 34
nvmlUnitGetCount, 34
nvmlUnitGetDevices, 35
nvmlUnitGetFanSpeedInfo, 35
nvmlUnitGetHandleByIndex, 35
nvmlUnitGetLedState, 36
nvmlUnitGetPsuInfo, 36
nvmlUnitGetTemperature, 37
nvmlUnitGetUnitInfo, 37
nvmlUnitSetLedState
nvmlUnitCommands, 69
nvmlUnitStructs
NVML_FAN_FAILED, 21
NVML_FAN_NORMAL, 21
NVML_LED_COLOR_AMBER, 21
NVML_LED_COLOR_GREEN, 21
nvmlFanState_t, 21
nvmlLedColor_t, 21
nvmlUtilization_t, 95

System Queries, 32

Unit Commands, 69
Unit Queries, 34
Unit Structs, 21
Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, the NVIDIA logo, GeForce, Tesla, and Quadro are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2007-2012 NVIDIA Corporation. All rights reserved.