# Contents

1 NVML API Reference .......................... 1
   1.1 Feature Matrix .......................... 4

2 Known issues in the current version of NVML library .......... 9

3 Change log of NVML library .................................. 11
   3.1 Changes between NVML v4.304 RC and v4.304 Production ........ 12
   3.2 Changes between NVML v4.304 RC and v4.304 Production ........ 12
   3.3 Changes between NVML v2.285 and v3.295 ................... 12
   3.4 Changes between NVML v1.0 and v2.285 ................... 13

4 Deprecated List ........................................ 15

5 Module Index ........................................... 17
   5.1 Modules ........................................... 17

6 Data Structure Index .................................... 19
   6.1 Data Structures .................................... 19

7 Module Documentation ...................................... 21
   7.1 Device Structs ..................................... 21
      7.1.1 Define Documentation .......................... 21
         7.1.1.1 NVML\_VALUE\_NOT\_AVAILABLE ............... 21
   7.2 Device Enums ....................................... 22
      7.2.1 Define Documentation .......................... 24
         7.2.1.1 NVML\_DOUBLE\_BIT\_ECC ................ 24
         7.2.1.2 NVML\_SINGLE\_BIT\_ECC ................ 24
         7.2.1.3 nvmlEccBitType_t .................... 24
      7.2.2 Enumeration Type Documentation ................... 24
         7.2.2.1 nvmlClockType_t ....................... 24
         7.2.2.2 nvmlComputeMode_t ................... 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.2.3</td>
<td>nvmlDriverModel_t</td>
<td>25</td>
</tr>
<tr>
<td>7.2.2.4</td>
<td>nvmlEccCounterType_t</td>
<td>25</td>
</tr>
<tr>
<td>7.2.2.5</td>
<td>nvmlEnableState_t</td>
<td>25</td>
</tr>
<tr>
<td>7.2.2.6</td>
<td>nvmlGpuOperationMode_t</td>
<td>26</td>
</tr>
<tr>
<td>7.2.2.7</td>
<td>nvmlInforomObject_t</td>
<td>26</td>
</tr>
<tr>
<td>7.2.2.8</td>
<td>nvmlMemoryErrorType_t</td>
<td>26</td>
</tr>
<tr>
<td>7.2.2.9</td>
<td>nvmlMemoryLocation_t</td>
<td>26</td>
</tr>
<tr>
<td>7.2.2.10</td>
<td>nvmlPstates_t</td>
<td>27</td>
</tr>
<tr>
<td>7.2.2.11</td>
<td>nvmlReturn_t</td>
<td>27</td>
</tr>
<tr>
<td>7.2.2.12</td>
<td>nvmlTemperatureSensors_t</td>
<td>28</td>
</tr>
<tr>
<td>7.3</td>
<td>Unit Structs</td>
<td>29</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Enumeration Type Documentation</td>
<td>29</td>
</tr>
<tr>
<td>7.3.1.1</td>
<td>nvmlFanState_t</td>
<td>29</td>
</tr>
<tr>
<td>7.3.1.2</td>
<td>nvmlLedColor_t</td>
<td>29</td>
</tr>
<tr>
<td>7.4</td>
<td>Event Types</td>
<td>30</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Detailed Description</td>
<td>30</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Define Documentation</td>
<td>30</td>
</tr>
<tr>
<td>7.4.2.1</td>
<td>nvmlEventTypeClock</td>
<td>30</td>
</tr>
<tr>
<td>7.4.2.2</td>
<td>nvmlEventTypePState</td>
<td>30</td>
</tr>
<tr>
<td>7.5</td>
<td>Initialization and Cleanup</td>
<td>31</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Detailed Description</td>
<td>31</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Function Documentation</td>
<td>31</td>
</tr>
<tr>
<td>7.5.2.1</td>
<td>nvmlInit</td>
<td>31</td>
</tr>
<tr>
<td>7.5.2.2</td>
<td>nvmlShutdown</td>
<td>31</td>
</tr>
<tr>
<td>7.6</td>
<td>Error reporting</td>
<td>32</td>
</tr>
<tr>
<td>7.6.1</td>
<td>Detailed Description</td>
<td>32</td>
</tr>
<tr>
<td>7.6.2</td>
<td>Function Documentation</td>
<td>32</td>
</tr>
<tr>
<td>7.6.2.1</td>
<td>nvmlErrorString</td>
<td>32</td>
</tr>
<tr>
<td>7.7</td>
<td>Constants</td>
<td>33</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Define Documentation</td>
<td>33</td>
</tr>
<tr>
<td>7.7.1.1</td>
<td>NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE</td>
<td>33</td>
</tr>
<tr>
<td>7.7.1.2</td>
<td>NVML_DEVICE_NAME_BUFFER_SIZE</td>
<td>33</td>
</tr>
<tr>
<td>7.7.1.3</td>
<td>NVML_DEVICE_SERIAL_BUFFER_SIZE</td>
<td>33</td>
</tr>
<tr>
<td>7.7.1.4</td>
<td>NVML_DEVICE_UUID_BUFFER_SIZE</td>
<td>33</td>
</tr>
<tr>
<td>7.7.1.5</td>
<td>NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE</td>
<td>33</td>
</tr>
<tr>
<td>7.7.1.6</td>
<td>NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE</td>
<td>33</td>
</tr>
<tr>
<td>7.7.1.7</td>
<td>NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE</td>
<td>33</td>
</tr>
</tbody>
</table>
7.8 System Queries

7.8.1 Detailed Description

7.8.2 Function Documentation

7.8.2.1 nvmlSystemGetDriverVersion

7.8.2.2 nvmlSystemGetNVMLVersion

7.8.2.3 nvmlSystemGetProcessName

7.9 Unit Queries

7.9.1 Detailed Description

7.9.2 Function Documentation

7.9.2.1 nvmlSystemGetHicVersion

7.9.2.2 nvmlUnitGetCount

7.9.2.3 nvmlUnitGetDevices

7.9.2.4 nvmlUnitGetFanSpeedInfo

7.9.2.5 nvmlUnitGetHandleByIndex

7.9.2.6 nvmlUnitGetLedState

7.9.2.7 nvmlUnitGetPsuInfo

7.9.2.8 nvmlUnitGetTemperature

7.9.2.9 nvmlUnitGetUnitInfo

7.10 Device Queries

7.10.1 Detailed Description

7.10.2 Function Documentation

7.10.2.1 nvmlDeviceGetApplicationsClock

7.10.2.2 nvmlDeviceGetClockInfo

7.10.2.3 nvmlDeviceGetComputeMode

7.10.2.4 nvmlDeviceGetComputeRunningProcesses

7.10.2.5 nvmlDeviceGetCount

7.10.2.6 nvmlDeviceGetCurrentClocksThrottleReasons

7.10.2.7 nvmlDeviceGetCurrPcieLinkGeneration

7.10.2.8 nvmlDeviceGetCurrPcieLinkWidth

7.10.2.9 nvmlDeviceGetDetailedEccErrors

7.10.2.10 nvmlDeviceGetDisplayMode

7.10.2.11 nvmlDeviceGetDriverModel

7.10.2.12 nvmlDeviceGetEccMode

7.10.2.13 nvmlDeviceGetFanSpeed

7.10.2.14 nvmlDeviceGetGpuOperationMode

7.10.2.15 nvmlDeviceGetHandleByIndex

7.10.2.16 nvmlDeviceGetHandleByPciBusId
<table>
<thead>
<tr>
<th>Section</th>
<th>Function Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.10.17</td>
<td>nvmlDeviceGetHandleBySerial</td>
<td>49</td>
</tr>
<tr>
<td>7.10.18</td>
<td>nvmlDeviceGetHandleByUUID</td>
<td>50</td>
</tr>
<tr>
<td>7.10.19</td>
<td>nvmlDeviceGetInforomConfigurationChecksum</td>
<td>50</td>
</tr>
<tr>
<td>7.10.20</td>
<td>nvmlDeviceGetInforomImageVersion</td>
<td>51</td>
</tr>
<tr>
<td>7.10.21</td>
<td>nvmlDeviceGetInforomVersion</td>
<td>51</td>
</tr>
<tr>
<td>7.10.22</td>
<td>nvmlDeviceGetMaxClockInfo</td>
<td>52</td>
</tr>
<tr>
<td>7.10.23</td>
<td>nvmlDeviceGetMaxPcieLinkGeneration</td>
<td>53</td>
</tr>
<tr>
<td>7.10.24</td>
<td>nvmlDeviceGetMaxPcieLinkWidth</td>
<td>53</td>
</tr>
<tr>
<td>7.10.25</td>
<td>nvmlDeviceGetMemoryErrorCounter</td>
<td>53</td>
</tr>
<tr>
<td>7.10.26</td>
<td>nvmlDeviceGetMemoryInfo</td>
<td>54</td>
</tr>
<tr>
<td>7.10.27</td>
<td>nvmlDeviceGetName</td>
<td>55</td>
</tr>
<tr>
<td>7.10.28</td>
<td>nvmlDeviceGetPciInfo</td>
<td>55</td>
</tr>
<tr>
<td>7.10.29</td>
<td>nvmlDeviceGetPerformanceState</td>
<td>55</td>
</tr>
<tr>
<td>7.10.30</td>
<td>nvmlDeviceGetPersistenceMode</td>
<td>56</td>
</tr>
<tr>
<td>7.10.31</td>
<td>nvmlDeviceGetPowerManagementDefaultLimit</td>
<td>56</td>
</tr>
<tr>
<td>7.10.32</td>
<td>nvmlDeviceGetPowerManagementLimit</td>
<td>57</td>
</tr>
<tr>
<td>7.10.33</td>
<td>nvmlDeviceGetPowerManagementLimitConstraints</td>
<td>57</td>
</tr>
<tr>
<td>7.10.34</td>
<td>nvmlDeviceGetPowerManagementMode</td>
<td>58</td>
</tr>
<tr>
<td>7.10.35</td>
<td>nvmlDeviceGetPowerState</td>
<td>59</td>
</tr>
<tr>
<td>7.10.36</td>
<td>nvmlDeviceGetPowerUsage</td>
<td>59</td>
</tr>
<tr>
<td>7.10.37</td>
<td>nvmlDeviceGetSerial</td>
<td>60</td>
</tr>
<tr>
<td>7.10.38</td>
<td>nvmlDeviceGetSupportedClocksThrottleReasons</td>
<td>60</td>
</tr>
<tr>
<td>7.10.39</td>
<td>nvmlDeviceGetSupportedGraphicsClocks</td>
<td>61</td>
</tr>
<tr>
<td>7.10.40</td>
<td>nvmlDeviceGetSupportedMemoryClocks</td>
<td>61</td>
</tr>
<tr>
<td>7.10.41</td>
<td>nvmlDeviceGetTemperature</td>
<td>62</td>
</tr>
<tr>
<td>7.10.42</td>
<td>nvmlDeviceGetTotalEccErrors</td>
<td>62</td>
</tr>
<tr>
<td>7.10.43</td>
<td>nvmlDeviceGetUtilizationRates</td>
<td>63</td>
</tr>
<tr>
<td>7.10.44</td>
<td>nvmlDeviceGetUUID</td>
<td>63</td>
</tr>
<tr>
<td>7.10.45</td>
<td>nvmlDeviceGetVbiosVersion</td>
<td>64</td>
</tr>
<tr>
<td>7.10.46</td>
<td>nvmlDeviceOnSameBoard</td>
<td>64</td>
</tr>
<tr>
<td>7.10.47</td>
<td>nvmlDeviceResetApplicationsClocks</td>
<td>65</td>
</tr>
<tr>
<td>7.10.48</td>
<td>nvmlDeviceValidateInforom</td>
<td>65</td>
</tr>
<tr>
<td>7.11 Unit Commands</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>7.11.1 Detailed Description</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>7.11.2 Function Documentation</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>7.11.2.1 nvmlUnitSetLedState</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>7.12 Device Commands</td>
<td></td>
<td>67</td>
</tr>
</tbody>
</table>
Chapter 1

NVML API Reference

The NVIDIA Management Library (NVML) is a C-based programmatic interface for monitoring and managing various states within NVIDIA Tesla ™ GPUs.

It is intended to be a platform for building 3rd party applications, and is also the underlying library for the NVIDIA-supported nvidia-smi tool.

NVML is thread-safe so it is safe to make simultaneous NVML calls from multiple threads.

API Documentation

Supported OS platforms:

- Windows: Windows Server 2008 R2 64bit, Windows 7 64bit
- Linux: 32-bit and 64-bit

Supported products:

- Full Support
  - NVIDIA Tesla ™ Line: S2050, C2050, C2070, C2075, M2050, M2070, M2075, M2090, X2070, X2090, K10, K20, K20X
  - NVIDIA Quadro ® Line: 4000, 5000, 6000, 7000, M2070-Q, 600, 2000, 3000M and 410
  - NVIDIA GeForce ® Line: None

- Limited Support
  - NVIDIA Tesla ™ Line: S1070, C1060, M1060
  - NVIDIA Quadro ® Line: All other current and previous generation Quadro-branded parts
  - NVIDIA GeForce ® Line: All current and previous generation GeForce-branded parts

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

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

The NVML API is divided into five categories:
• Support Methods:
  – Initialization and Cleanup

• Query Methods:
  – System Queries
  – Device Queries
  – Unit Queries

• Control Methods:
  – Unit Commands
  – Device Commands

• Event Handling Methods:
  – Event Handling Methods

• Error reporting Methods
  – Error reporting

List of changes can be found in the Changelog
## 1.1 Feature Matrix

<table>
<thead>
<tr>
<th>Queries</th>
<th>C2190</th>
<th>C2170</th>
<th>M2050</th>
<th>M2070</th>
<th>M2070</th>
<th>M2090</th>
<th>M2090</th>
<th>S2190</th>
<th>K2070</th>
<th>K2090</th>
<th>K10</th>
<th>K20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board Serial Number</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPU UUID</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>BIOS Version</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>InForen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Version</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OEM Object Version</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ECC Object Version</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Power Object Version</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Checksum</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PC Info</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Compute Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Display Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Persistence Mode (Linux-Only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ECC Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ECC Error Counter Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device Memory</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Register File</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L1 Cache</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Texture Memory</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Driver Model (Windows-Only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fan Speed</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPU Temperature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Power Readings</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Management Enabled</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Current Power Draw</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Power Draw Limit</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Default Power Limit</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Power Limit Constraints</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Clock Speeds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphics</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Application Clocks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphics</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphics</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SM</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Listing Supported Clock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combinations</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Utilization Rates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECC Errors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volatile</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Location-Based</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Total</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Aggregate</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Location-Based</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Total</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Performance State</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Process Info</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPU Operation Mode (GDM)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 1.1: This chart shows which features are reported for each Fermi and Kepler architecture GPU product.
<table>
<thead>
<tr>
<th>Queries</th>
<th>4000</th>
<th>5000</th>
<th>6000</th>
<th>Quadro</th>
<th>Quadro</th>
<th>P2000</th>
<th>M2070</th>
<th>6000</th>
<th>2000</th>
<th>3000</th>
<th>410</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Serial Number</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>GPU UUID</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>BIOS Version</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Image Version</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>OEM Object Version</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>ECC Object Version</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Power Object Version</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Checksum</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>PCI Info</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Compute Mode</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Display Mode</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Persistence Mode (Linux-Only)</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>ECC Mode</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>ECC Error Counter Type</td>
<td>Device Memory</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Register File</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>L1 Cache</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>L2 Cache</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Texture Memory</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Driver Model (Windows-Only)</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Fan Speed</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>GPU Temperature</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Memory Usage</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Power Readings</td>
<td>Power Management Enabled</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Current Power Draw</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Power Draw Limit</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Default Power Limit</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Power Limit Constraints</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Clock Speeds</td>
<td>Current</td>
<td>Graphics</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>SM</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Application Clocks</td>
<td>Graphics</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Mix</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Graphics</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>SM</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Listing Supported Clock combinations</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Utilization Rates</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>ECC Errors</td>
<td>Volatile Location-Based</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Aggregate</td>
<td>Location-Based</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Performance State</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Process Info</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>GPU Operation Mode (GDM)</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>

Figure 1.2: This chart shows which features are reported for each Quadro and T10 GPU product.
Figure 1.3: This chart shows which commands are available for each Fermi and Kepler architecture GPU product.

<table>
<thead>
<tr>
<th>Commands</th>
<th>C2050</th>
<th>C2070</th>
<th>C2075</th>
<th>M2050</th>
<th>M2070</th>
<th>M2075</th>
<th>M2090</th>
<th>S2050</th>
<th>X2070</th>
<th>X2090</th>
<th>K10</th>
<th>K20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Compute Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set Persistence Mode (Linux-Only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set Display Model (Win7-Only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set ECC Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Clear ECC Errors</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reset GPU (Linux-Only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set Application Clocks</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set Power Management Limit</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set GPU Operation Mode (GOM)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 1.4: This chart shows which commands are available for each Quadro and T10 GPU product.

<table>
<thead>
<tr>
<th>Commands</th>
<th>4000</th>
<th>5000</th>
<th>6000</th>
<th>Quadro</th>
<th>Plex 7000</th>
<th>M2070-Q</th>
<th>600</th>
<th>2000</th>
<th>3000M</th>
<th>410</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Compute Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set Persistence Mode (Linux-Only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set Display Model (Win7-Only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set ECC Mode</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Clear ECC Errors</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reset GPU (Linux-Only)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set Application Clocks</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set Power Management Limit</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Set GPU Operation Mode (GOM)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Figure 1.5: This chart shows which unit-level features are available for each S-class product. All GPUs within each S-class product also provide the information listed in the Device chart below.
Chapter 2

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_t::usedGpuMemory` value that is larger than the actual value. This will be fixed in a future release.
- On Linux GPU Reset can’t be triggered when there is pending GPU Operation Mode (GOM) change
- On Linux GPU Reset may not successfully change pending ECC mode. A full reboot may be required to enable the mode change.
Chapter 3

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

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

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

- Added `nvmlDeviceGetGpuOperationMode` and `nvmlDeviceSetGpuOperationMode`

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

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

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

### 3.3 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`
3.4 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`. 

- 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
Chapter 4

Depreciated 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_TYPEUNCORRECTED`

Global `NVML_SINGLE_BIT_ECC` Mapped to `NVML_MEMORY_ERROR_TYPECORRECTED`

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`. 
Chapter 5

Module Index

5.1 Modules

Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Structs</td>
<td>21</td>
</tr>
<tr>
<td>Device Enums</td>
<td>22</td>
</tr>
<tr>
<td>Unit Structs</td>
<td>29</td>
</tr>
<tr>
<td>Initialization and Cleanup</td>
<td>31</td>
</tr>
<tr>
<td>Error reporting</td>
<td>32</td>
</tr>
<tr>
<td>Constants</td>
<td>33</td>
</tr>
<tr>
<td>System Queries</td>
<td>34</td>
</tr>
<tr>
<td>Unit Queries</td>
<td>36</td>
</tr>
<tr>
<td>Device Queries</td>
<td>40</td>
</tr>
<tr>
<td>Unit Commands</td>
<td>66</td>
</tr>
<tr>
<td>Device Commands</td>
<td>67</td>
</tr>
<tr>
<td>Event Handling Methods</td>
<td>73</td>
</tr>
<tr>
<td>Event Types</td>
<td>30</td>
</tr>
<tr>
<td>NvmlClocksThrottleReasons</td>
<td>77</td>
</tr>
</tbody>
</table>
Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

- `nvmlEccErrorCounts_t` .................................................. 79
- `nvmlEventData_t` ......................................................... 80
- `nvmlHwbcEntry_t` ......................................................... 81
- `nvmlLedState_t` .......................................................... 82
- `nvmlMemory_t` ........................................................... 83
- `nvmlPciInfo_t` ........................................................... 84
- `nvmlProcessInfo_t` ...................................................... 85
- `nvmlPSUInfo_t` .......................................................... 86
- `nvmlUnitFanInfo_t` ...................................................... 87
- `nvmlUnitFanSpeeds_t` ................................................. 88
- `nvmlUnitInfo_t` .......................................................... 89
- `nvmlUtilization_t` ....................................................... 90
Chapter 7

Module Documentation

7.1 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)

7.1.1 Define Documentation

7.1.1.1 #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.
7.2 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,
    NVML_MEMORY_ERROR_TYPE_COUNT
}`

- `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,
    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_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
}

7.2.1 Define Documentation

7.2.1.1 #define NVML_DOUBLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_UNCORRECTED

Double bit ECC errors

Deprecated

    Mapped to NVML_MEMORY_ERROR_TYPE_UNCORRECTED

7.2.1.2 #define NVML_SINGLE_BIT_ECC NVML_MEMORY_ERROR_TYPE_CORRECTED

Single bit ECC errors

Deprecated

    Mapped to NVML_MEMORY_ERROR_TYPE_CORRECTED

7.2.1.3 #define nvmlEccBitType_t nvmlMemoryErrorType_t

ECC bit types.

Deprecated

    See nvmlMemoryErrorType_t for a more flexible type

7.2.2 Enumeration Type Documentation

7.2.2.1 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.
7.2 Device Enums

7.2.2.2 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.

7.2.2.3 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.

7.2.2.4 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).

7.2.2.5 enum nvmlEnableState_t

Generic enable/disable enum.

** Enumerator:**

- **NVML_FEATURE_DISABLED** Feature disabled.
- **NVML_FEATURE_ENABLED** Feature enabled.
7.2.2.6 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.

7.2.2.7 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.

7.2.2.8 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.

7.2.2.9 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.
7.2 Device Enums

7.2.2.10 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_14` Performance state 14.
- `NVML_PSTATE_15` Performance state 15 – Minimum Performance.
- `NVML_PSTATE_UNKNOWN` Unknown performance state.

7.2.2.11 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_UNKNOWN`  An internal driver error occurred.
7.2.2.12  enum nvmlTemperatureSensors_t

Temperature sensors.

Enumerator:

`NVML_TEMPERATURE_GPU` Temperature sensor for the GPU die.
7.3 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
}

7.3.1 Enumeration Type Documentation

7.3.1.1 enum nvmlFanState_t

Fan state enum.

Enumerator:

- NVML_FAN_NORMAL  Fan is working properly.
- NVML_FAN_FAILED  Fan has failed.

7.3.1.2 enum nvmlLedColor_t

Led color enum.

Enumerator:

- NVML_LED_COLOR_GREEN  GREEN, indicates good health.
- NVML_LED_COLOR_AMBER  AMBER, indicates problem.
### 7.4 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.

#### 7.4.1 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`.

#### 7.4.2 Define Documentation

##### 7.4.2.1 `#define nvmlEventTypeClock 0x0000000000000010LL`

Kepler only

##### 7.4.2.2 `#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.
7.5 Initialization and Cleanup

Functions

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

7.5.1 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.

7.5.2 Function Documentation

7.5.2.1 `nvmlReturn_t DECLDIR nvmlInit (void)`

Initialize NVML by discovering and attaching to all GPU devices in the system.
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_NO_PERMISSION` if the user doesn’t have permission to talk to any device
- `NVML_ERROR_DRIVER_NOT_LOADED` if NVIDIA driver is not running
- `NVML_ERROR_INSUFFICIENT_POWER` if any devices have improperly attached external power cables
- `NVML_ERROR_UNKNOWN` on any unexpected error

7.5.2.2 `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()`.

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
7.6 Error reporting

Functions

• const DECLDIR char * nvmlErrorString(nvmlReturn_t result)

7.6.1 Detailed Description

This chapter describes helper functions for error reporting routines.

7.6.2 Function Documentation

7.6.2.1 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.
# 7.7 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`

## 7.7.1 Define Documentation

### 7.7.1.1 `#define NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE 16`

Buffer size guaranteed to be large enough for `nvmlDeviceGetInforomVersion` and `nvmlDeviceGetInforomImageVersion`

### 7.7.1.2 `#define NVML_DEVICE_NAME_BUFFER_SIZE 64`

Buffer size guaranteed to be large enough for `nvmlDeviceGetName`

### 7.7.1.3 `#define NVML_DEVICE_SERIAL_BUFFER_SIZE 30`

Buffer size guaranteed to be large enough for `nvmlDeviceGetSerial`

### 7.7.1.4 `#define NVML_DEVICE_UUID_BUFFER_SIZE 80`

Buffer size guaranteed to be large enough for `nvmlDeviceGetUUID`

### 7.7.1.5 `#define NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE 32`

Buffer size guaranteed to be large enough for `nvmlDeviceGetVbiosVersion`

### 7.7.1.6 `#define NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE 80`

Buffer size guaranteed to be large enough for `nvmlSystemGetDriverVersion`

### 7.7.1.7 `#define NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE 80`

Buffer size guaranteed to be large enough for `nvmlSystemGetNVMLVersion`
7.8 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)`

7.8.1 Detailed Description

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

7.8.2 Function Documentation

7.8.2.1 `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

7.8.2.2 `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
7.8 System Queries

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

7.8.2.3 `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
7.9 Unit Queries

Functions

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

7.9.1 Detailed Description

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

7.9.2 Function Documentation

7.9.2.1 `nvmlReturn_t DECLDIR nvmlSystemGetHicVersion (unsigned int * hwbcCount, nvmlHwbcEntry_t * hwbcEntries)`

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

For S-class products.

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

Parameters:

- `hwbcCount` Size of hwbcEntries array
- `hwbcEntries` Array holding information about hwbc

Returns:

- `NVML_SUCCESS` if `hwbcCount` and `hwbcEntries` have been populated
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_INVALID_ARGUMENT` if either `hwbcCount` or `hwbcEntries` is NULL
- `NVML_ERROR_INSUFFICIENT_SIZE` if `hwbcCount` indicates that the `hwbcEntries` array is too small

7.9.2.2 `nvmlReturn_t DECLDIR nvmlUnitGetCount (unsigned int * unitCount)`

Retrieves the number of units in the system.

For S-class products.
7.9 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

7.9.2.3 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

7.9.2.4 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
7.9.2.5  

```
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

7.9.2.6

```
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()`

7.9.2.7

```
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.
7.9 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

7.9.2.8 `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

7.9.2.9 `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
7.10 Device Queries

Functions

- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetCount (unsigned int *deviceCount)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetHandleByIndex (unsigned int index, nvmlDevice_t *device)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetHandleBySerial (const char *serial, nvmlDevice_t *device)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetHandleByUUID (const char *uuid, nvmlDevice_t *device)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetHandleByPciBusId (const char *pciBusId, nvmlDevice_t *device)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetName (nvmlDevice_t device, char *name, unsigned int length)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetSerial (nvmlDevice_t device, char *serial, unsigned int length)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetUUID (nvmlDevice_t device, char *uid, unsigned int length)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetInforomVersion (nvmlDevice_t device, nvmlInforomObject_t object, char *version, unsigned int length)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetInforomImageVersion (nvmlDevice_t device, char *version, unsigned int length)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetInforomConfigurationChecksum (nvmlDevice_t device, unsigned int *checksum)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceValidateInforom (nvmlDevice_t device)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetDisplayMode (nvmlDevice_t device, nvmlEnableState_t *display)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetPersistenceMode (nvmlDevice_t device, nvmlEnableState_t *mode)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetPciInfo (nvmlDevice_t device, nvmlPciInfo_t *pci)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkGeneration (nvmlDevice_t device, unsigned int *maxLinkGen)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetMaxPcieLinkWidth (nvmlDevice_t device, unsigned int *maxLinkWidth)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkGeneration (nvmlDevice_t device, unsigned int *currLinkGen)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetCurrPcieLinkWidth (nvmlDevice_t device, unsigned int *currLinkWidth)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetMaxClockInfo (nvmlDevice_t device, nvmlClockType_t type, unsigned int *clock)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetApplicationsClock (nvmlDevice_t device, nvmlClockType_t clockType, unsigned int *clockMHz)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceResetApplicationsClocks (nvmlDevice_t device)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetSupportedMemoryClocks (nvmlDevice_t device, unsigned int *count, unsigned int *clocksMHz)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetSupportedGraphicsClocks (nvmlDevice_t device, unsigned int memoryClockMHz, unsigned int *count, unsigned int *clocksMHz)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetFanSpeed (nvmlDevice_t device, unsigned int *speed)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetTemperature (nvmlDevice_t device, nvmlTemperatureSensors_t sensorType, unsigned int *temp)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetPerformanceState (nvmlDevice_t device, nvmlPstates_t *pState)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetCurrentClocksThrottleReasons (nvmlDevice_t device, unsigned long long *clocksThrottleReasons)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetSupportedClocksThrottleReasons (nvmlDevice_t device, unsigned long long supportedClocksThrottleReasons)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetPowerState (nvmlDevice_t device, nvmlPstates_t *pState)}
- \texttt{nvmiReturn_t DECLDIR nvmlDeviceGetPowerManagementMode (nvmlDevice_t device, nvmlEnableState_t *mode)}
7.10 Device Queries

- 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)

7.10.1 Detailed Description

This chapter describes 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() or nvmlDeviceGetHandleByPciBusId().

7.10.2 Function Documentation

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

Retrieves the clock that applications will use unless an overspec situation occurs. Can be changed using nvmlDeviceSetApplicationsClocks.

For Tesla™ products, and Quadro ® 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 new settings were successfully set
- NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
- NVML_ERROR_NOT_FOUND if the max clock limit is not set
- NVML_ERROR_INVALID_ARGUMENT if device is invalid or clockMHz is NULL or clockType is invalid
- NVML_ERROR_NOT_SUPPORTED if the device cannot report the specified clock
- NVML_ERROR_UNKNOWN on any unexpected error

7.10.2.2 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_UNKNOWN on any unexpected error

7.10.2.3 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.

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_UNKNOWN on any unexpected error

See also:
   nvmlDeviceSetComputeMode()

7.10.2.4 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_UNKNOWN on any unexpected error

See also:
   nvmlSystemGetProcessName

7.10.2.5 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.
On some platforms not all devices may be accessible due to permission restrictions. In these cases the device count
will reflect only the GPUs that NVML can access.

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

NVIDIA Management Library
Returns:

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

### 7.10.2.6 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:**

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

**Returns:**

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

**See also:**

- NvmlClocksThrottleReasons
- nvmlDeviceGetSupportedClocksThrottleReasons

### 7.10.2.7 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 max 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_UNKNOWN** on any unexpected error
7.10 Device Queries

7.10.2.8 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:
- \textit{device}  The identifier of the target device
- \textit{currLinkWidth}  Reference in which to return the max PCIe link generation

Returns:
- \textbf{NVML\_SUCCESS} if \textit{currLinkWidth} has been populated
- \textbf{NVML\_ERROR\_UNINITIALIZED} if the library has not been successfully initialized
- \textbf{NVML\_ERROR\_INVALID\_ARGUMENT} if \textit{device} is invalid or \textit{currLinkWidth} is null
- \textbf{NVML\_ERROR\_NOT\_SUPPORTED} if PCIe link information is not available
- \textbf{NVML\_ERROR\_UNKNOWN} on any unexpected error

7.10.2.9 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.

\textbf{Deprecated}

This API supports only a fixed set of ECC error locations On different GPU architectures different locations are supported See \textit{nvmlDeviceGetMemoryErrorCounter}

For Tesla™ and Quadro ® products from the Fermi and Kepler families. Requires \textit{NVML\_INFOROM\_ECC} version 2.0 or higher to report aggregate location-based ECC counts. Requires \textit{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 \textit{nvmlMemoryErrorType_t} for a description of available bit types.
See \textit{nvmlEccCounterType_t} for a description of available counter types.
See \textit{nvmlEccErrorCounts_t} for a description of provided detailed ECC counts.

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

Returns:
- \textbf{NVML\_SUCCESS} if \textit{eccCounts} has been populated
• NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
• NVML_ERROR_INVALID_ARGUMENT if device, errorType or counterType is invalid, or eccCounts is NULL
• NVML_ERROR_NOT_SUPPORTED if the device does not support this feature
• NVML_ERROR_UNKNOWN on any unexpected error

See also:

nvmlDeviceClearEccErrorCounts()
• 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_UNKNOWN on any unexpected error

See also:
	nvmlDeviceSetDriverModel()
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_UNKNOWN on any unexpected error

7.10.2.14 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.

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_UNKNOWN on any unexpected error

See also:

- nvmlGpuOperationMode_t
- nvmlDeviceSetGpuOperationMode

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

Acquire the handle for a particular device, based on its index.
For all products.
Valid indices are derived from the accessibleDevices count returned by nvmlDeviceGetCount(). 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 board serial numbers. See nvmlDeviceGetHandleBySerial() and nvmlDeviceGetHandleByPciBusId().
7.10 Device Queries

Parameters:

- **index** The index of the target GPU, \( \geq 0 \) 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_UNKNOWN** on any unexpected error

7.10.2.16 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().

Parameters:

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

Returns:

- **NVML_SUCCESS** if **device** has been set
- **NVML_ERROR_UNINITIALIZED** if the library has not been successfully initialized
- **NVML_ERROR_INVALID_ARGUMENT** if **pciBusId** is invalid or **device** is NULL
- **NVML_ERROR_NOT_FOUND** if **pciBusId** does not match a valid device on the system
- **NVML_ERROR_UNKNOWN** on any unexpected error

7.10.2.17 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.

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_UNKNOWN** on any unexpected error

See also:

`nvmlDeviceGetSerial`
`nvmlDeviceGetHandleByUUID`

7.10.2.18  `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

**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_UNKNOWN** on any unexpected error

See also:

`nvmlDeviceGetUUID`

7.10.2.19  `nvmlReturn_t DECLDIR nvmlDeviceGetInf ormConfigurationChecksum (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
7.10 Device Queries

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_UNKNOWN on any unexpected error

7.10.2.20  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
- NVML_ERROR_UNKNOWN on any unexpected error

See also:

nvmlDeviceGetInforomVersion

7.10.2.21  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_UNKNOWN on any unexpected error

See also:

nvmlDeviceGetInforomImageVersion

---

`7.10.2.22` 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.

**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_UNKNOWN on any unexpected error
7.10 Device Queries

7.10.2.23 `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_UNKNOWN` on any unexpected error

7.10.2.24 `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_UNKNOWN` on any unexpected error

7.10.2.25 `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.

NVIDIA Management Library
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_UNKNOWN on any unexpected error

7.10.2.26 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_UNKNOWN on any unexpected error
7.10 Device Queries

7.10.2.27  

```
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

7.10.2.28  

```
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_UNKNOWN` on any unexpected error

7.10.2.29  

```
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_NOT_SUPPORTED** if the device does not support this feature
- **NVML_ERROR_UNKNOWN** on any unexpected error

7.10.2.30  `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_UNKNOWN** on any unexpected error

See also:

- `nvmlDeviceSetPersistenceMode()`

7.10.2.31  `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_ERRORUNKNOWN** on any unexpected error

### 7.10.2.32 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.

- 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_ERRORUNKNOWN** on any unexpected error

### 7.10.2.33 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` has 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_UNKNOWN** on any unexpected error

See also:

nvmlDeviceSetPowerManagementLimit

7.10.2.34 _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_UNKNOWN** on any unexpected error
7.10.2.35  `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_UNKNOWN` on any unexpected error

---

7.10.2.36  `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_UNKNOWN` on any unexpected error
7.10.2.37  

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

7.10.2.38  

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 returned successfully
• 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_UNKNOWN on any unexpected error

See also:

NvmlClocksThrottleReasons
nvmlDeviceGetCurrentClocksThrottleReasons
7.10.2.39  

```c
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 new settings were successfully set
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_NOT_FOUND` if the max clock limit is not set
- `NVML_ERROR_INVALID_ARGUMENT` if `device` is invalid or `clock` is NULL or `clockType` is invalid
- `NVML_ERROR_NOT_SUPPORTED` if the device cannot report the specified clock
- `NVML_ERROR_INSUFFICIENT_SIZE` if `count` is too small
- `NVML_ERROR_UNKNOWN` on any unexpected error

**See also:**

- `nvmlDeviceSetApplicationsClocks`
- `nvmlDeviceGetSupportedMemoryClocks`

7.10.2.40  

```c
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, and Quadro ® 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 new settings were successfully set
- `NVML_ERROR_UNINITIALIZED` if the library has not been successfully initialized
- `NVML_ERROR_NOT_FOUND` if the max clock limit is not set
- `NVML_ERROR_INVALID_ARGUMENT` if `device` is invalid or `clock` is NULL or `clockType` is invalid
- `NVML_ERROR_NOT_SUPPORTED` if the device cannot report the specified clock
- `NVML_ERROR_INSUFFICIENT_SIZE` if `count` is too small
- `NVML_ERROR_UNKNOWN` on any unexpected error
See also:

nvmlDeviceSetApplicationsClocks
nvmlDeviceGetSupportedGraphicsClocks

7.10.2.41 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_UNKNOWN on any unexpected error

7.10.2.42 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
7.10 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_UNKNOWN on any unexpected error

See also:

nvmlDeviceClearEccErrorCounts()

7.10.2.43 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.

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_UNKNOWN on any unexpected error

7.10.2.44 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

NVIDIA Management Library
• 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_UNKNOWN on any unexpected error

7.10.2.45 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_UNKNOWN on any unexpected error

7.10.2.46 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 when onSameBoard has been populated
• NVML_ERROR_UNINITIALIZED if the library has not been successfully initialized
• NVML_ERROR_INVALID_ARGUMENT if dev1, dev2 or onSameBoard are invalid
• NVML_ERROR_UNKNOWN on any unexpected error
7.10.2.47 `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`
- `nvmlDeviceSetApplicationsClock`

For Tesla™ products, and Quadro® 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_NOT_FOUND` if the max clock limit is not set
- `NVML_ERROR_INVALID_ARGUMENT` if `device` is invalid
- `NVML_ERROR_NOT_SUPPORTED` if the device cannot perform this action
- `NVML_ERROR_UNKNOWN` on any unexpected error

7.10.2.48 `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_UNKNOWN` on any unexpected error
7.11 Unit Commands

Functions

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

7.11.1 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.

7.11.2 Function Documentation

7.11.2.1 `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()`
### 7.12 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)`

#### 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.

#### Function Documentation

##### 7.12.2.1 nvmlReturn_t DECLDIR nvmlDeviceClearEccErrorCounts (nvmlDevice_t device, nvmlEccCounterType_t counterType)

Clear the ECC 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 `nvmlEccCounterType_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_UNKNOWN` on any unexpected error

NVIDIA Management Library
See also:
  • nvmlDeviceGetDetailedEccErrors()
  • nvmlDeviceGetTotalEccErrors()

7.12.2.2 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, and Quadro ® 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.

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_UNKNOWN  on any unexpected error

7.12.2.3 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.
### 7.12 Device Commands

#### 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_UNKNOWN on any unexpected error

See also:

nvmlDeviceGetComputeMode()

#### 7.12.2.4 nvmlReturn_t DECLDIR nvmlDeviceSetDriverModel (nvmlDevice_t device, nvmlDriverModel_t driverModel, unsigned int flags)

Set the driver model for the device.

For Tesla ™ 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

#### 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

NVIDIA Management Library
• NVML_ERROR_NO_PERMISSION if the user doesn’t have permission to perform this operation
• NVML_ERROR_UNKNOWN on any unexpected error

See also:

nvmlDeviceGetDriverModel()

7.12.2.5 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
  ecc  The target ECC mode

Returns:

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

See also:

nvmlDeviceGetEccMode()

7.12.2.6 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 is 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_UNKNOWN on any unexpected error

See also:

- nvmlGpuOperationMode_t
- nvmlDeviceGetGpuOperationMode

7.12.2.7  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_UNKNOWN on any unexpected error

See also:

- nvmlDeviceGetPersistenceMode()
7.12.2.8  

**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_UNKNOWN** on any unexpected error

See also:
- `nvmlDeviceGetPowerManagementLimitConstraints`
- `nvmlDeviceGetPowerManagementDefaultLimit`
7.13 Event Handling Methods

Data Structures

• struct nvmlEventData_t

Modules

• Event Types

Typedefs

• typedef struct nvmlEventSet_st *nvmlEventSet_t

Functions

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

7.13.1 Detailed Description

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

7.13.2 Typedef Documentation

7.13.2.1 typedef struct nvmlEventSet_st * nvmlEventSet_t

Handle to an event set

7.13.3 Function Documentation

7.13.3.1 nvmlReturn_t DECLDIR nvmlDeviceGetSupportedEventTypes (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
**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_UNKNOWN** on any unexpected error

**See also:**

- Event Types
- `nvmlDeviceRegisterEvents`

### 7.13.3.2 `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`

**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_UNKNOWN** on any unexpected error

**See also:**

- Event Types
- `nvmlDeviceGetSupportedEventTypes`
- `nvmlEventSetWait`
- `nvmlEventSetFree`
7.13.3.3 \texttt{nvmlReturn_t DECLDIR nvmlEventSetCreate (nvmlEventSet_t * set)}

Create an empty set of events. Event set should be freed by \texttt{nvmlEventSetFree}

\textbf{Parameters:}

\begin{itemize}
  \item \texttt{set} Reference in which to return the event handle
\end{itemize}

\textbf{Returns:}

\begin{itemize}
  \item \texttt{NVML\_SUCCESS} if the event has been set
  \item \texttt{NVML\_ERROR\_UNINITIALIZED} if the library has not been successfully initialized
  \item \texttt{NVML\_ERROR\_INVALID\_ARGUMENT} if \texttt{set} is NULL
  \item \texttt{NVML\_ERROR\_UNKNOWN} on any unexpected error
\end{itemize}

\textbf{See also:}

\texttt{nvmlEventSetFree}

7.13.3.4 \texttt{nvmlReturn_t DECLDIR nvmlEventSetFree (nvmlEventSet_t set)}

Releases events in the set

For Tesla \textsuperscript{TM} and Quadro \textsuperscript{®} products from the Fermi and Kepler families.

\textbf{Parameters:}

\begin{itemize}
  \item \texttt{set} Reference to events to be released
\end{itemize}

\textbf{Returns:}

\begin{itemize}
  \item \texttt{NVML\_SUCCESS} if the event has been successfully released
  \item \texttt{NVML\_ERROR\_UNINITIALIZED} if the library has not been successfully initialized
  \item \texttt{NVML\_ERROR\_UNKNOWN} on any unexpected error
\end{itemize}

\textbf{See also:}

\texttt{nvmlDeviceRegisterEvents}

7.13.3.5 \texttt{nvmlReturn_t DECLDIR nvmlEventSetWait (nvmlEventSet_t set, nvmlEventData_t * data, unsigned int timeoutms)}

Waits on events and delivers events

For Tesla \textsuperscript{TM} and Quadro \textsuperscript{®} 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)

\textbf{Parameters:}

\begin{itemize}
  \item \texttt{set} Reference to set of events to wait on
  \item \texttt{data} Reference in which to return event data
\end{itemize}
**timeouts**  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_UNKNOWN` on any unexpected error

**See also:**

- Event Types
- `nvmlDeviceRegisterEvents`
7.14 NvmlClocksThrottleReasons

Defines

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

7.14.1 Define Documentation

7.14.1.1 #define nvmlClocksThrottleReasonAll

Value:

```
(nvmlClocksThrottleReasonNone \n | nvmlClocksThrottleReasonGpuIdle \n | nvmlClocksThrottleReasonUserDefinedClocks \n | nvmlClocksThrottleReasonSwPowerCap \n | nvmlClocksThrottleReasonHwSlowdown \n | nvmlClocksThrottleReasonUnknown)```

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

7.14.1.2 #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.

7.14.1.3 #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
### Module Documentation

7.14.1.4  #define nvmlClocksThrottleReasonNone 0x0000000000000000LL

Bit mask representing no clocks throttling

Clocks are as high as possible.

7.14.1.5  #define nvmlClocksThrottleReasonSwPowerCap 0x0000000000000004LL

SW Power Scaling algorithm is reducing the clocks below requested clocks

See also:
- `nvmlDeviceGetPowerUsage`
- `nvmlDeviceSetPowerManagementLimit`
- `nvmlDeviceGetPowerManagementLimit`

7.14.1.6  #define nvmlClocksThrottleReasonUnknown 0x8000000000000000LL

Some other unspecified factor is reducing the clocks

7.14.1.7  #define nvmlClocksThrottleReasonUserDefinedClocks 0x0000000000000002LL

GPU clocks are limited by user specified limit

See also:
- `nvmlDeviceSetApplicationsClocks`
- `nvmlDeviceGetApplicationsClock`
Chapter 8

Data Structure Documentation

8.1 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.

8.1.1 Detailed Description

Detailed ECC error counts for a device.

Deprecated

Different GPU families can have different memory error counters. See nvmlDeviceGetMemoryErrorCounter
8.2  nvmlEventData_t Struct Reference

#include <nvml.h>

Data Fields

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

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

8.2.1  Detailed Description

Information about occurred event
8.3 nvmlHwbcEntry_t Struct Reference

#include <nvml.h>

8.3.1 Detailed Description

Description of HWBC entry
8.4 nvmlLedState_t Struct Reference

#include <nvml.h>

Data Fields

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

- nvmlLedColor_t color
  
  GREEN or AMBER.

8.4.1 Detailed Description

LED states for an S-class unit.
#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.*

## 8.5.1 Detailed Description

Memory allocation information for a device.
Data Structure Documentation

8.6  `nvmlPciInfo_t` Struct Reference

#include <nvml.h>

Data Fields

- **char busId [16]**
  
  *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.*

8.6.1 Detailed Description

PCI information about a GPU device.
# 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.

## 8.7.1 Detailed Description

Information about running compute processes on the GPU
8.8 nvmlPSUInfo_t Struct Reference

#include <nvml.h>

Data Fields

- char state [256]
  
  The power supply state.

- unsigned int current
  
  PSU current (A).

- unsigned int voltage
  
  PSU voltage (V).

- unsigned int power
  
  PSU power draw (W).

8.8.1 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
#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.

## 8.9.1 Detailed Description

Fan speed reading for a single fan in an S-class unit.
8.10  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.

8.10.1  Detailed Description

Fan speed readings for an entire S-class unit.
#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.*

**8.11.1 Detailed Description**

Static S-class unit info.
8.12  `nvmlUtilization_t` Struct Reference

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

**Data Fields**

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

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

### 8.12.1 Detailed Description

Utilization information for a device.
Index

Constants, 33
Device Commands, 67
Device Enums, 22
Device Queries, 40
Device Structs, 21

Error reporting, 32
Event Handling Methods, 73
Event Types, 30

Initialization and Cleanup, 31

NVML_AGGREGATE_ECC
nvmlDeviceEnumvs, 25
NVML_CLOCK_GRAPHICS
  nvmlDeviceEnumvs, 24
NVML_CLOCK_MEM
  nvmlDeviceEnumvs, 24
NVML_CLOCK_SM
  nvmlDeviceEnumvs, 24
NVML_COMPUTEMODE_DEFAULT
  nvmlDeviceEnumvs, 25
NVML_COMPUTEMODE_EXCLUSIVE_PROCESS
  nvmlDeviceEnumvs, 25
NVML_COMPUTEMODE_EXCLUSIVE_THREAD
  nvmlDeviceEnumvs, 25
NVML_COMPUTEMODE_PROHIBITED
  nvmlDeviceEnumvs, 25
NVML_DRIVER_WDDM
  nvmlDeviceEnumvs, 25
NVML_DRIVER_WDM
  nvmlDeviceEnumvs, 25
NVML_ERROR_ALREADY_INITIALIZED
  nvmlDeviceEnumvs, 27
NVML_ERROR_CORRUPTED_INFOROM
  nvmlDeviceEnumvs, 27
NVML_ERROR_DRIVER_NOT_LOADED
  nvmlDeviceEnumvs, 27
NVML_ERROR_FUNCTION_NOT_FOUND
  nvmlDeviceEnumvs, 27
NVML_ERROR_INSUFFICIENT_POWER
  nvmlDeviceEnumvs, 27
NVML_ERROR_INSUFFICIENT_SIZE
  nvmlDeviceEnumvs, 27
NVML_ERROR_INVALID_ARGUMENT
  nvmlDeviceEnumvs, 27
NVML_ERROR_IRQ_ISSUE
  nvmlDeviceEnumvs, 27
NVML_ERROR_LIBRARY_NOT_FOUND
  nvmlDeviceEnumvs, 27
NVML_ERROR_NO_PERMISSION
  nvmlDeviceEnumvs, 27
NVML_ERROR_NOT_FOUND
  nvmlDeviceEnumvs, 27
NVML_ERROR_NOT_SUPPORTED
  nvmlDeviceEnumvs, 27
NVML_ERROR_TIMEOUT
  nvmlDeviceEnumvs, 27
NVML_ERROR_UNINITIALIZED
  nvmlDeviceEnumvs, 27
NVML_ERROR_UNKNOWN
  nvmlDeviceEnumvs, 27

NVML_ERROR_DEVICE_FUNCTION_NOT_FOUND
  nvmlDeviceEnumvs, 27

NVML_FAN_FAILED
  nvmlUnitStructs, 29
NVML_FAN_NORMAL
  nvmlUnitStructs, 29

NVML_FEATURE_DISABLED
  nvmlDeviceEnumvs, 25
NVML_FEATURE_ENABLED
  nvmlDeviceEnumvs, 25

NVML_INFOROM_COUNT
  nvmlDeviceEnumvs, 26
NVML_INFOROM_ECC
  nvmlDeviceEnumvs, 26
NVML_INFOROM_OEM
  nvmlDeviceEnumvs, 26
NVML_INFOROM_POWER
  nvmlDeviceEnumvs, 26

NVML_LED_COLOR_AMBER
  nvmlUnitStructs, 29
NVML_LED_COLOR_GREEN
  nvmlUnitStructs, 29

NVML_MEMORY_ERROR_TYPE_CORRECTED
  nvmlDeviceEnumvs, 26
NVML_MEMORY_ERROR_TYPE_COUNT
nvmlDeviceEnumvs, 26

NVML_MEMORY_ERROR_TYPE_UNCORRECTED
nvmlDeviceEnumvs, 26

NVML_MEMORY_LOCATION_COUNT
nvmlDeviceEnumvs, 26

NVML_MEMORY_LOCATION_DEVICE_MEMORY
nvmlDeviceEnumvs, 26

NVML_MEMORY_LOCATION_L1_CACHE
nvmlDeviceEnumvs, 26

NVML_MEMORY_LOCATION_L2_CACHE
nvmlDeviceEnumvs, 26

NVML_MEMORY_LOCATION_REGISTER_FILE
nvmlDeviceEnumvs, 26

NVML_MEMORY_LOCATION_TEXTURE_MEMORY
nvmlDeviceEnumvs, 26

NVML_PSTATE_0
nvmlDeviceEnumvs, 27

NVML_PSTATE_1
nvmlDeviceEnumvs, 27

NVML_PSTATE_10
nvmlDeviceEnumvs, 27

NVML_PSTATE_11
nvmlDeviceEnumvs, 27

NVML_PSTATE_12
nvmlDeviceEnumvs, 27

NVML_PSTATE_13
nvmlDevice Enumvs, 27

NVML_PSTATE_14
nvmlDeviceEnumvs, 27

NVML_PSTATE_15
nvmlDeviceEnumvs, 27

NVML_PSTATE_2
nvmlDeviceEnumvs, 27

NVML_PSTATE_3
nvmlDeviceEnumvs, 27

NVML_PSTATE_4
nvmlDeviceEnumvs, 27

NVML_PSTATE_5
nvmlDeviceEnumvs, 27

NVML_PSTATE_6
nvmlDeviceEnumvs, 27

NVML_PSTATE_7
nvmlDeviceEnumvs, 27

NVML_PSTATE_8
nvmlDeviceEnumvs, 27

NVML_PSTATE_9
nvmlDeviceEnumvs, 27

NVML_PSTATE_UNKNOWN
nvmlDeviceEnumvs, 27

NVML_PSTATE_10
nvmlDeviceEnumvs, 27

NVML_PSTATE_11
nvmlDeviceEnumvs, 27

NVML_PSTATE_12
nvmlDeviceEnumvs, 27

NVML_PSTATE_13
nvmlDeviceEnumvs, 27

NVML_PSTATE_14
nvmlDeviceEnumvs, 27

NVML_PSTATE_15
nvmlDeviceEnumvs, 27

NVML_PSTATE_2
nvmlDeviceEnumvs, 27

NVML_PSTATE_3
nvmlDeviceEnumvs, 27

NVML_PSTATE_4
nvmlDeviceEnumvs, 27

NVML_PSTATE_5
nvmlDeviceEnumvs, 27

NVML_PSTATE_6
nvmlDeviceEnumvs, 27

NVML_PSTATE_7
nvmlDeviceEnumvs, 27

NVML_PSTATE_8
nvmlDeviceEnumvs, 27

NVML_PSTATE_9
nvmlDeviceEnumvs, 27

NVML_PSTATE_UNKNOWN
nvmlDeviceEnumvs, 27

NVML_SUCCESS
nvmlDeviceEnumvs, 27

NVML_TEMPERATURE_GPU
nvmlDeviceEnumvs, 28

NVML_VOLATILE_ECC
nvmlDeviceEnumvs, 25

NVML_DEVICE_INFOROM_VERSION_BUFFER_SIZE
nvmlConstants, 33

NVML_DEVICE_NAME_BUFFER_SIZE
nvmlConstants, 33

NVML_DEVICE_SERIAL_BUFFER_SIZE
nvmlConstants, 33

NVML_DEVICE_UUID_BUFFER_SIZE
nvmlConstants, 33

NVML_DEVICE_VBIOS_VERSION_BUFFER_SIZE
nvmlConstants, 33

NVML_DOUBLE_BIT_ECC
nvmlDeviceEnumvs, 24

NVML_SINGLE_BIT_ECC
nvmlDeviceEnumvs, 24

NVML_SYSTEM_DRIVER_VERSION_BUFFER_SIZE
nvmlConstants, 33

NVML_SYSTEM_NVML_VERSION_BUFFER_SIZE
nvmlConstants, 33

NVML_VALUE_NOT_AVAILABLE
nvmlDeviceStructs, 21

nvmlClocksThrottleReasonAll
nvmlClocksThrottleReasons, 77

nvmlClocksThrottleReasonGpuIdle
nvmlClocksThrottleReasons, 77

nvmlClocksThrottleReasonHwSlowdown
nvmlClocksThrottleReasons, 77

nvmlClocksThrottleReasonNone
nvmlClocksThrottleReasons, 77

nvmlClocksThrottleReasonSwPowerCap
nvmlClocksThrottleReasons, 78

nvmlClocksThrottleReasonUnknown
nvmlClocksThrottleReasons, 78

nvmlClocksThrottleReasonUserDefinedClocks
nvmlClocksThrottleReasons, 78

nvmlClocksThrottleReasonSwPowerCap
nvmlClocksThrottleReasons, 78

nvmlClocksThrottleReasonUnknown
nvmlClocksThrottleReasons, 78

nvmlClocksThrottleReasonUserDefinedClocks
nvmlClocksThrottleReasons, 78

nvmlClockType_t
nvmlDeviceEnumvs, 24

cnvmlComputeMode_t
nvmlDeviceEnumvs, 24

nvmlConstants

NVIDIA Management Library
| NVML DEVICE INFOROM_VERSION_BUFFER_SIZE, 33 | NVML GOM LOW DP, 26 |
| NVML DEVICE NAME_BUFFER_SIZE, 33 | NVML INFOROM_COUNT, 26 |
| NVML DEVICE SERIAL_BUFFER_SIZE, 33 | NVML INFOROM ECC, 26 |
| NVML DEVICE_UUID_BUFFER_SIZE, 33 | NVML INFOROM OEM, 26 |
| NVML DEVICE VBIOS_VERSION_BUFFER_SIZE, 33 | NVML INFOROM POWER, 26 |
| NVML SYSTEM DRIVER_VERSION_BUFFER_SIZE, 33 | NVML MEMORY ERROR_TYPE_CORRECTED, 26 |
| NVML SYSTEM NVML_VERSION_BUFFER_SIZE, 33 | NVML MEMORY ERROR_TYPE_COUNT, 26 |
| nvmlDeviceClearEccErrorCounts | NVML MEMORY ERROR_TYPE_UNCORRECTED, 26 |
| nvmlDeviceCommands, 67 | NVML MEMORY LOCATION_COUNT, 26 |
| nvmlDeviceCommands | NVML MEMORY LOCATION_DEVICE_MEMORY, 26 |
| nvmlDeviceClearEccErrorCounts, 67 | NVML_MEMORY_LOCATION_L1_CACHE, 26 |
| nvmlDeviceSetApplicationsClocks, 68 | NVML_MEMORY_LOCATION_L2_CACHE, 26 |
| nvmlDeviceSetComputeMode, 68 | NVML_MEMORY_LOCATION_REGISTER_FILE, 26 |
| nvmlDeviceSetDriverModel, 69 | NVML_MEMORY_LOCATION_TEXTURE_MEMORY, 26 |
| nvmlDeviceSetEccMode, 70 | nvmlClockType_t, 24 |
| nvmlDeviceSetGpuOperationMode, 70 | nvmlComputeMode_t, 24 |
| nvmlDeviceSetPersistenceMode, 71 | nvmlDriverModel_t, 25 |
| nvmlDeviceSetPowerManagementLimit, 71 | nvmlEccBitType_t, 24 |
| nvmlDeviceEnumvs | nvmlEccCounterType_t, 25 |
| NVML AGGREGATE ECC, 25 | nvmlEnableState_t, 25 |
| NVML CLOCK_GRAPHICS, 24 | nvmlGpuOperationMode_t, 25 |
| NVML CLOCK_MEM, 24 | nvmlInforomObject_t, 26 |
| NVML CLOCK SM, 24 | nvmlMemoryErrorType_t, 26 |
| NVML COMPUTE MODE DEFAULT, 25 | nvmlMemoryLocation_t, 26 |
| NVML COMPUTE MODE EXCLUSIVE, 25 | nvmlPstates_t, 26 |
| nvml Compute Mode, 25 | nvmlReturn_t, 27 |
| nvmlGpuOperationMode, 70 | nvmlTemperatureSensors_t, 27 |
nvmlDeviceGetApplicationsClock
    nvmlDeviceQueries, 41
nvmlDeviceGetClockInfo
    nvmlDeviceQueries, 42
nvmlDeviceGetComputeMode
    nvmlDeviceQueries, 42
nvmlDeviceGetComputeRunningProcesses
    nvmlDeviceQueries, 43
nvmlDeviceGetCount
    nvmlDeviceQueries, 43
nvmlDeviceGetCurrentClocksThrottleReasons
    nvmlDeviceQueries, 44
nvmlDeviceGetCurrPcieLinkGeneration
    nvmlDeviceQueries, 44
nvmlDeviceGetCurrPcieLinkWidth
    nvmlDeviceQueries, 44
nvmlDeviceGetDetailedEccErrors
    nvmlDeviceQueries, 45
nvmlDeviceGetDisplayMode
    nvmlDeviceQueries, 46
nvmlDeviceGetDriverModel
    nvmlDeviceQueries, 46
nvmlDeviceGetEccMode
    nvmlDeviceQueries, 47
nvmlDeviceGetFanSpeed
    nvmlDeviceQueries, 47
nvmlDeviceGetGpuOperationMode
    nvmlDeviceQueries, 48
nvmlDeviceGetHandleByIndex
    nvmlDeviceQueries, 48
nvmlDeviceGetHandleByPciBusId
    nvmlDeviceQueries, 49
nvmlDeviceGetHandleBySerial
    nvmlDeviceQueries, 49
nvmlDeviceGetHandleByUUID
    nvmlDeviceQueries, 50
nvmlDeviceGetInforomConfigurationChecksum
    nvmlDeviceQueries, 50
nvmlDeviceGetInforomImageVersion
    nvmlDeviceQueries, 51
nvmlDeviceGetInforomVersion
    nvmlDeviceQueries, 51
nvmlDeviceGetMaxClockInfo
    nvmlDeviceQueries, 52
nvmlDeviceGetMaxPcieLinkGeneration
    nvmlDeviceQueries, 52
nvmlDeviceGetMaxPcieLinkWidth
    nvmlDeviceQueries, 53
nvmlDeviceGetMemoryErrorCounter
    nvmlDeviceQueries, 53
nvmlDeviceGetMemoryInfo
    nvmlDeviceQueries, 54
nvmlDeviceGetName
    nvmlDeviceQueries, 54
nvmlDeviceGetPciInfo
    nvmlDeviceQueries, 55
nvmlDeviceGetPerformanceState
    nvmlDeviceQueries, 55
nvmlDeviceGetPersistenceMode
    nvmlDeviceQueries, 56
nvmlDeviceGetPowerManagementDefaultLimit
    nvmlDeviceQueries, 56
nvmlDeviceGetPowerManagementLimit
    nvmlDeviceQueries, 57
nvmlDeviceGetPowerManagementLimitConstraints
    nvmlDeviceQueries, 57
nvmlDeviceGetPowerManagementMode
    nvmlDeviceQueries, 58
nvmlDeviceGetPowerState
    nvmlDeviceQueries, 58
nvmlDeviceGetPowerUsage
    nvmlDeviceQueries, 59
nvmlDeviceGetSerial
    nvmlDeviceQueries, 59
nvmlDeviceGetSupportedClocksThrottleReasons
    nvmlDeviceQueries, 60
nvmlDeviceGetSupportedEventTypes
    nvmlEvents, 73
nvmlDeviceGetSupportedGraphicsClocks
    nvmlDeviceQueries, 60
nvmlDeviceGetSupportedMemoryClocks
    nvmlDeviceQueries, 61
nvmlDeviceGetTemperature
    nvmlDeviceQueries, 62
nvmlDeviceGetTotalEccErrors
    nvmlDeviceQueries, 62
nvmlDeviceGetUtilizationRates
    nvmlDeviceQueries, 63
nvmlDeviceGetVbiosVersion
    nvmlDeviceQueries, 63
nvmlDeviceOnSameBoard
    nvmlDeviceQueries, 64
nvmlDeviceQueries
    nvmlDeviceGetApplicationsClock, 41
    nvmlDeviceGetClockInfo, 42
    nvmlDeviceGetComputeMode, 42
    nvmlDeviceGetComputeRunningProcesses, 43
    nvmlDeviceGetCount, 43
    nvmlDeviceGetCurrentClocksThrottleReasons, 44
    nvmlDeviceGetCurrPcieLinkGeneration, 44
    nvmlDeviceGetCurrPcieLinkWidth, 44
    nvmlDeviceGetDetailedEccErrors, 45
    nvmlDeviceGetDisplayMode, 46
    nvmlDeviceGetDriverModel, 46
    nvmlDeviceGetEccMode, 47
    nvmlDeviceGetFanSpeed, 47
    nvmlDeviceQueries, 55
    nvmlDeviceGetPerformanceState
    nvmlDeviceQueries, 55
    nvmlDeviceGetPersistenceMode
    nvmlDeviceQueries, 56
    nvmlDeviceGetPowerManagementDefaultLimit
    nvmlDeviceQueries, 56
    nvmlDeviceGetPowerManagementLimit
    nvmlDeviceQueries, 57
    nvmlDeviceGetPowerManagementLimitConstraints
    nvmlDeviceQueries, 57
    nvmlDeviceGetPowerManagementMode
    nvmlDeviceQueries, 58
    nvmlDeviceGetPowerState
    nvmlDeviceQueries, 58
    nvmlDeviceGetPowerUsage
    nvmlDeviceQueries, 59
    nvmlDeviceGetSerial
    nvmlDeviceQueries, 59
    nvmlDeviceGetSupportedClocksThrottleReasons
    nvmlDeviceQueries, 60
    nvmlDeviceGetSupportedEventTypes
    nvmlEvents, 73
    nvmlDeviceGetSupportedGraphicsClocks
    nvmlDeviceQueries, 60
    nvmlDeviceGetSupportedMemoryClocks
    nvmlDeviceQueries, 61
    nvmlDeviceGetTemperature
    nvmlDeviceQueries, 62
    nvmlDeviceGetTotalEccErrors
    nvmlDeviceQueries, 62
    nvmlDeviceGetUtilizationRates
    nvmlDeviceQueries, 63
    nvmlDeviceGetVbiosVersion
    nvmlDeviceQueries, 63
    nvmlDeviceOnSameBoard
    nvmlDeviceQueries, 64
    nvmlDeviceQueries
    nvmlDeviceGetApplicationsClock, 41
    nvmlDeviceGetClockInfo, 42
    nvmlDeviceGetComputeMode, 42
    nvmlDeviceGetComputeRunningProcesses, 43
    nvmlDeviceGetCount, 43
    nvmlDeviceGetCurrentClocksThrottleReasons, 44
    nvmlDeviceGetCurrPcieLinkGeneration, 44
    nvmlDeviceGetCurrPcieLinkWidth, 44
    nvmlDeviceGetDetailedEccErrors, 45
    nvmlDeviceGetDisplayMode, 46
    nvmlDeviceGetDriverModel, 46
    nvmlDeviceGetEccMode, 47
    nvmlDeviceGetFanSpeed, 47
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvmlDeviceGetGpuOperationMode</td>
<td>48</td>
</tr>
<tr>
<td>nvmlDeviceGetHandleByIndex</td>
<td>48</td>
</tr>
<tr>
<td>nvmlDeviceGetHandleByPciBusId</td>
<td>49</td>
</tr>
<tr>
<td>nvmlDeviceGetHandleBySerial</td>
<td>49</td>
</tr>
<tr>
<td>nvmlDeviceGetHandleByUUID</td>
<td>50</td>
</tr>
<tr>
<td>nvmlDeviceGetInforomConfigurationChecksum</td>
<td>50</td>
</tr>
<tr>
<td>nvmlDeviceGetInforomImageVersion</td>
<td>51</td>
</tr>
<tr>
<td>nvmlDeviceGetInforomVersion</td>
<td>51</td>
</tr>
<tr>
<td>nvmlDeviceGetMaxClockInfo</td>
<td>52</td>
</tr>
<tr>
<td>nvmlDeviceGetMaxPcieLinkGeneration</td>
<td>52</td>
</tr>
<tr>
<td>nvmlDeviceGetMaxPcieLinkWidth</td>
<td>53</td>
</tr>
<tr>
<td>nvmlDeviceGetMemoryErrorCounter</td>
<td>53</td>
</tr>
<tr>
<td>nvmlDeviceGetMemoryInfo</td>
<td>54</td>
</tr>
<tr>
<td>nvmlDeviceGetName</td>
<td>54</td>
</tr>
<tr>
<td>nvmlDeviceGetPciInfo</td>
<td>55</td>
</tr>
<tr>
<td>nvmlDeviceGetPerformanceState</td>
<td>55</td>
</tr>
<tr>
<td>nvmlDeviceGetPersistencesMode</td>
<td>56</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerManagementDefaultLimit</td>
<td>56</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerManagementLimit</td>
<td>57</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerManagementLimitConstraints</td>
<td>57</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerManagementMode</td>
<td>58</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerState</td>
<td>58</td>
</tr>
<tr>
<td>nvmlDeviceGetPowerUsage</td>
<td>59</td>
</tr>
<tr>
<td>nvmlDeviceGetSerial</td>
<td>59</td>
</tr>
<tr>
<td>nvmlDeviceGetSupportedClocksThrottleReasons</td>
<td>60</td>
</tr>
<tr>
<td>nvmlDeviceGetSupportedGraphicsClocks</td>
<td>60</td>
</tr>
<tr>
<td>nvmlDeviceGetSupportedMemoryClocks</td>
<td>61</td>
</tr>
<tr>
<td>nvmlDeviceGetTemperature</td>
<td>62</td>
</tr>
<tr>
<td>nvmlDeviceGetTotalEccErrors</td>
<td>62</td>
</tr>
<tr>
<td>nvmlDeviceGetUtilizationRates</td>
<td>63</td>
</tr>
<tr>
<td>nvmlDeviceGetUUID</td>
<td>63</td>
</tr>
<tr>
<td>nvmlDeviceGetVbiosVersion</td>
<td>64</td>
</tr>
<tr>
<td>nvmlDeviceGetOnSameBoard</td>
<td>64</td>
</tr>
<tr>
<td>nvmlDeviceResetApplicationsClocks</td>
<td>64</td>
</tr>
<tr>
<td>nvmlDeviceValidateInforom</td>
<td>65</td>
</tr>
<tr>
<td>nvmlDeviceRegisterEvents</td>
<td>74</td>
</tr>
<tr>
<td>nvmlDeviceResetApplicationsClocks</td>
<td>74</td>
</tr>
<tr>
<td>nvmlDeviceQueries</td>
<td>64</td>
</tr>
<tr>
<td>nvmlDeviceCommands</td>
<td>71</td>
</tr>
<tr>
<td>nvmlDeviceStructs</td>
<td>21</td>
</tr>
<tr>
<td>NVML_VALUE_NOT_AVAILABLE</td>
<td>21</td>
</tr>
<tr>
<td>nvmlDeviceValidateInforom</td>
<td>65</td>
</tr>
<tr>
<td>nvmlDeviceQueries</td>
<td>65</td>
</tr>
<tr>
<td>nvmlDriverModel_t</td>
<td>25</td>
</tr>
<tr>
<td>nvmlDeviceEnvs</td>
<td>25</td>
</tr>
<tr>
<td>nvmlDriverModel_t</td>
<td>24</td>
</tr>
<tr>
<td>nvmlDeviceEnvs</td>
<td>25</td>
</tr>
<tr>
<td>nvmlDeviceEnvs</td>
<td>25</td>
</tr>
<tr>
<td>nvmlDeviceEnvs</td>
<td>79</td>
</tr>
<tr>
<td>nvmlEnableState_t</td>
<td>79</td>
</tr>
<tr>
<td>nvmlEvents</td>
<td>80</td>
</tr>
<tr>
<td>nvmlEventData_t</td>
<td>73</td>
</tr>
<tr>
<td>nvmlEvents</td>
<td>73</td>
</tr>
<tr>
<td>nvmlEventSetCreate</td>
<td>74</td>
</tr>
<tr>
<td>nvmlEvents</td>
<td>74</td>
</tr>
<tr>
<td>nvmlEventSetFree</td>
<td>75</td>
</tr>
<tr>
<td>nvmlEvents</td>
<td>75</td>
</tr>
<tr>
<td>nvmlEventSetWait</td>
<td>75</td>
</tr>
<tr>
<td>nvmlEventSet_t</td>
<td>75</td>
</tr>
<tr>
<td>nvmlEventSet_t</td>
<td>73</td>
</tr>
<tr>
<td>nvmlEventSetCreate</td>
<td>74</td>
</tr>
<tr>
<td>nvmlEventSetFree</td>
<td>75</td>
</tr>
<tr>
<td>nvmlEventSetWait</td>
<td>75</td>
</tr>
<tr>
<td>nvmlEventType</td>
<td>75</td>
</tr>
<tr>
<td>nvmlEventTypeClock</td>
<td>30</td>
</tr>
<tr>
<td>nvmlEventTypePState</td>
<td>30</td>
</tr>
<tr>
<td>nvmlEventTypeClock</td>
<td>30</td>
</tr>
<tr>
<td>nvmlEventTypePState</td>
<td>30</td>
</tr>
<tr>
<td>nvmlFanState_t</td>
<td>29</td>
</tr>
<tr>
<td>nvmlUnitStructs</td>
<td>29</td>
</tr>
<tr>
<td>nvmlGpuOperationMode_t</td>
<td>25</td>
</tr>
<tr>
<td>nvmlDeviceEnvs</td>
<td>25</td>
</tr>
<tr>
<td>nvmlHwbcEntry_t</td>
<td>81</td>
</tr>
<tr>
<td>nvmlInforomObject_t</td>
<td>26</td>
</tr>
<tr>
<td>nvmlInitializationAndCleanup</td>
<td>31</td>
</tr>
<tr>
<td>nvmlInitializationAndCleanup</td>
<td>31</td>
</tr>
<tr>
<td>nvmlInit</td>
<td>31</td>
</tr>
<tr>
<td>nvmlShutdown</td>
<td>31</td>
</tr>
<tr>
<td>nvmlLedColor_t</td>
<td>31</td>
</tr>
</tbody>
</table>

NVIDIA Management Library
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.