Class name: nvidiaapi

Header:

#include <QMutex>

#include <QLibrary>

#include <QByteArray>

#include <QThread>

#include "nvapi.h"

Name space: std

Description: nvidiaapi class provides the interface to access and set Nvidia GPU information.

Constructors:

**NvidiaAPI**();

Parameters: None.

Return: None.

Task: Initialize the Nvidia variables, special data structure for storing version, status, flags, and const parameter that will be used.

Destructor:

~ **NvidiaAPI** ():

Parameters: None.

Return: None.

Task: Free the necessary space used in the class.

Key Private Variables:

|  |  |
| --- | --- |
| NvAPI\_QueryInterface\_t NvQueryInterface; | Interface for query information in Nvidia |
| NvAPI\_Initialize\_t NvInit; | Initialize parameter of device to query Nvidia |
| NvAPI\_Unload\_t NvUnload; | Unload the data and connection from Nvidia API |
| NvAPI\_EnumPhysicalGPUs\_t NvEnumGPUs; | parameter for the enumerator of the GPU |
| NvAPI\_GPU\_GetSystemType\_t NvGetSysType; | parameter for the types of system of GPU |
| NvAPI\_GPU\_GetFullName\_t NvGetName; | parameter for the name of the GPU |
| NvAPI\_GPU\_GetPhysicalFrameBufferSize\_t NvGetMemSize; | parameter for the memory size of GPU |
| NvAPI\_GPU\_GetRamType\_t NvGetMemType; | parameter for the memory type of GPU |
| NvAPI\_GPU\_GetVbiosVersionString\_t NvGetBiosName; | parameter for the Bios name of GPU |
| NvAPI\_GPU\_GetAllClockFrequencies\_t NvGetFreq; | parameter for the frequency data of GPU |
| NvAPI\_GPU\_GetPstates20\_t NvGetPstates; | parameter for the get states of GPU |
| NvAPI\_GPU\_SetPstates20\_t NvSetPstates; | parameter for the set states of GPU |
| NvAPI\_GPU\_GetPstatesInfoEx\_t NvGetPStatesInfoEx; | parameter for the get statesEx of GPU |
| NvAPI\_GPU\_GetIllumination\_t NvGetIllumination; | parameter for the get illumination of GPU |
| NvAPI\_GPU\_SetIllumination\_t NvSetIllumination; | parameter for the set illumination of GPU |
| NvAPI\_GPU\_QueryIlluminationSupport\_t NvQueryIlluminationSupport; | parameter for the query illumination of GPU |
| NvAPI\_DLL\_ClientPowerPoliciesGetStatus\_t NvClientPowerPoliciesGetStatus; | parameter for the client to get status of GPU |
| NvAPI\_DLL\_ClientPowerPoliciesGetInfo\_t NvClientPowerPoliciesGetInfo; | parameter for the client to get info of GPU |
| NvAPI\_DLL\_ClientPowerPoliciesSetStatus\_t NvClientPowerPoliciesSetStatus; | parameter for the client to set status of GPU |
| NvAPI\_GPU\_GetCoolersSettings\_t NvGetCoolersSettings; | parameter to get cooler setting of GPU |
| NvAPI\_GPU\_SetCoolerLevel\_t NvSetCoolerLevel; | parameter to set cooler level of GPU |
| NvAPI\_GPU\_GetThermalSettings\_t NvGetThermalSettings; | parameter to get thermal setting of GPU |

Key Public Variables:

|  |  |
| --- | --- |
| NvU32 \_gpuCount; | Counts of GPU |
| int TempLimit; | Upper bound of temperature |
| NvPhysicalGpuHandle \_gpuHandles[NVAPI\_MAX\_PHYSICAL\_GPUS]; | Status set of gpu |
| bool \_libLoaded; | Indicator of if lib is loaded |
| fanSpeedThread\* \_fanThread; | Pointer to thread for running fan |

Public Methods:

unsigned int **getGPUCount**();

Parameters: None.

Return: None.

Task: get the number of GPU.

int **getGpuTemperature**(unsigned int gpu);

Parameters: GPU index.

Return: Temperature in Celsius degree.

Task: access sensor to get the temperature of target GPU device.

int **ControlGpuTemperature**(unsigned int gpu);

Parameters: GPU index.

Return: status of Nvidia API accessing (0 represents OK).

Task: A cooling down process. It reduces overclocking offset when the temperature so too high.

int **getGPUOffset**(unsigned int gpu);

Parameters: GPU index.

Return: GPU overclocking offset.

Task: access Nvidia API to query GPU overclocking offset.

int **getMemOffset**(unsigned int gpu);

Parameters: GPU index.

Return: Memory overclocking offset.

Task: access Nvidia API to query Memory overclocking offset.

int **getMemClock**(unsigned int gpu);

Parameters: GPU index.

Return: Memory clocking.

Task: access Nvidia API to query Memory clocking, which represents the running memory clocking.

unsigned int **getGpuClock**(unsigned int gpu);

Parameters: GPU index.

Return: GPU clocking.

Task: access Nvidia API to query GPU clocking, which represents the running GPU clocking.

unsigned int **getPowerLimit**(unsigned int gpu);

Parameters: GPU index.

Return: Power Limit.

Task: access Nvidia API to get the limit value of Power.

unsigned int **getFanSpeed**(unsigned int gpu);

Parameters: GPU index.

Return: Fan Speed.

Task: access Nvidia API to query hardware to get the speed of fan.

int **setMemClockOffset**(unsigned int gpu, int clock);

Parameters: GPU index, clock that should be set.

Return: status of Nvidia API accessing (0 represents OK).

Task: set Memory Overclocking offset so that memory clock limitation will be changed.

int **setGPUOffset**(unsigned int gpu, int offset);

Parameters: GPU index, clock that should be set.

Return: status of Nvidia API accessing (0 represents OK).

Task: set GPU Overclocking offset so that GPU clock limitation will be changed.

int **setPowerLimitPercent**(unsigned int gpu, unsigned int percent);

Parameters: GPU index, percent of limitation of power.

Return: status of Nvidia API accessing (0 represents OK).

Task: set Power Limit so that power used in mining will be limited in given percent value.

int **setTempLimitOffset**(unsigned int gpu, unsigned int offset);

Parameters: GPU index, temperature limit that should be set.

Return: status of Nvidia API accessing (0 represents OK).

Task: set the upper bound of temperature which gpu should not be higher than this.

int **getTempLimitOffset**(unsigned int gpu);

Parameters: GPU index

Return: upper bound of temperature which gpu should not be higher than this.

Task: query the temperature upper bound that has been set already.

int **setFanSpeed**(unsigned int gpu, unsigned int percent);

Parameters: GPU index, percent of max speed of fan.

Return: status of Nvidia API accessing (0 represents OK).

Task: set fan speed percent so that fan rotates in given percent value of maximum speed.

bool **libLoaded**()

Parameters: None.

Return: signal that represent if Nvidia lib is loaded.

Task: To check if the Nvidia lib is loaded and use in this file.

void **startFanThread**();

Parameters: None.

Return: None.

Task: start a thread for running fan rotating.

void **stopFanThread**();

Parameters: None.

Return: None.

Task: stop a thread for running fan rotating.