UTILITIES AND APIS

NvAPI Reference Manual

Version 6.0
NVIDIA Confidential
Released Under NDA

NVIDIA Corporation
October 27, 2006

Published by
NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

Table of Contents

Overview

NvAPI is an application programming interface (API) that provides direct access to many features of NVIDIA hardware that are not available through the operating system. Currently, the API focuses on graphics hardware, but is designed to provide support for all NVIDIA hardware in the future.

It is NVIDIA's goal to develop NvAPI with the following key design features:

- Uniform interface across Microsoft® Windows® XP, Linux, and Windows Vista
- Long term stability

  It is expected that NvAPI will provide a stable platform for programs using the interface– so that, for example, a program written today will still work three years from now on much newer NVIDIA hardware and drivers.

About This Document

This document is provided **under a non-disclosure agreement (NDA)** to OEMs, game developers, and NVIDIA technology partners.

This document describes the interface constants, structure definitions, and function prototypes for NvAPI.

Document Revision History

Revision	Date	Description
1.0	2/3/06	Initial Release
2.0	4/11/06	Added:
		NvAPI_SetRefreshRateOverride()
		NvAPI_OGL_ExpertModeGet() / NvAPI_OGL_ExpertModeSet()
		NvAPI_OGL_ExpertModeDefaultsSet() / NvAPI_OGL_ExpertModeDefaultsGet()
		NvAPI_I2CRead()
		NvAPI_I2CWrite()
3.0	5/8/06	Added:
		NvAPI_CreateDisplayFromUnattachedDisplay()
		NvAPI_EnumNvidiaUnAttachedDisplayHandle()
		NvAPI_GetPhysicalGPUFromUnattachedDisplay()
4.0	7/19/06	Added:
		Several GPU InformationCalls
		GPU Clock Control Calls section
		GPU Cooler Calls section
		Thermal API Calls section

Revision	Date	Description

5.0	9/22/06	Added:
		NvAPI_D3D9_GpuSyncGetHandleSize()
		NvAPI_D3D9_GpuSyncInit()
		NvAPI_D3D9_GpuSyncEnd()
		NvAPI_D3D9_GpuSyncMapTexBuffer()
		NvAPI_D3D9_GpuSyncMapVertexBuffer()
		NvAPI_D3D9_GpuSyncAcquire()
		NvAPI_D3D9_GpuSyncRelease()
		NvAPI_D3D9_PresentVideo()
		NvAPI_D3D9_SetResourceHint()
		NvAPI_D3D9_Lock()
		NvAPI_D3D9_Unlock()
		NvAPI_D3D9_LockForCUDA()
		NvAPI_GPU_GetFullName()
		NvAPI_GetSupportedViews()
		NvAPI_GetVBlankCounter()
		NvAPI_GetView()
		NvAPI_SetView()

6.0	10/27/06	Added:
		NvAPI_GetAssociatedDisplayOutputId())
		NvAPI_GPU_GetQuadroStatus()
		NvAPI_SetViewEx()
		NvAPI_GPU_GetConnectedOutputsWithLidState()
		NvAPI_GetHDMISupportInfo()
		NvAPI_GetInfoFrame()
		NvAPI_SetInfoFrame()

System Requirements

NvAPI Files

Make sure you have the following files:

- nvapi.h
- nvapi.lib (for 32-bit support)
- nvapi64.lib (for 64-bit support)

Operating System and Platforms

The current version of NvAPI is supported on Windows XP and Windows Vista, both 32-bit and 64-bit architectures. Some API calls are OS specific–see the individual calls for OS support information.

NVIDIA ForceWare Driver Version

- NvAPI is supported on ForceWare driver versions 81.20 and up.
- This document includes calls appearing in driver versions from 82.61 up to 96.60. See the individual calls for specific driver version information.

2

NvAPI

This chapter describes the NvAPI in the following sections:

Important NvAPI Concepts

NvAPI Handles

NvAPI handles are retrieved from various calls and passed to other calls in NvAPI. These are meant to be opaque types, and do not necessarily correspond to specific indices, HDCs, or display indices.

Most handles remain valid until a display re-configuration such as a display mode set, or a GPU reconfiguration such as going into or out of SLI modes. If NvAPI returns **`NVAPI_HANDLE_INVALIDATED`**, the application should discard all handles and re-enumerate them.

The following is a description of key NvAPI handles and identifiers:

- **Physical GPU handle (typedef void * NvPhysicalGpuHandle):** NvPhysicalGpuHandle is a reference to a physical GPU.

  Each GPU in a multi-GPU board will have its own handle. GPUs are assigned a handle even if they are not in use by the OS.
- **Logical GPU handle (typedef void * NvLogicalGpuHandle):** NvLogicalGpuHandle is a reference to one or more physical GPUs acting as a single logical device.

  A single GPU will have a single logical GPU handle and a single physical GPU handle. Two GPUs acting in an SLI configuration will have a single logical GPU handle and two physical GPU handles.
- **NVIDIA display handle (typedef void * NvDisplayHandle) :** NVIDIA display handles map one-to-one to the GDI handles for the attached displays in the Windows Display Properties Settings page.

  NvDisplayHandles reflect only the *displays* that the OS is aware of. Therefore, there is only one NvDisplayHandle for displays in Clone or Span mode, but there are two in Dualview mode.

  Some APIs use **NvUnAttachedDisplayHandle** for GDI dsplays that are not attached.
- **GPU output**: GPU output IDs are identifiers for the GPU outputs that drive display devices. The GPU output might or might not be connected to a display, or be active.

  Each output is identified by a bit setting within a 32-bit unsigned integer. A GPU output mask consists of a 32-bit integer with several bits set, identifying more than one output from the same physical GPU.

Figure 2.1, Figure 2.2, and Figure 2.3 illustrates these four identifiers used by NvAPI under various GPU configurations.

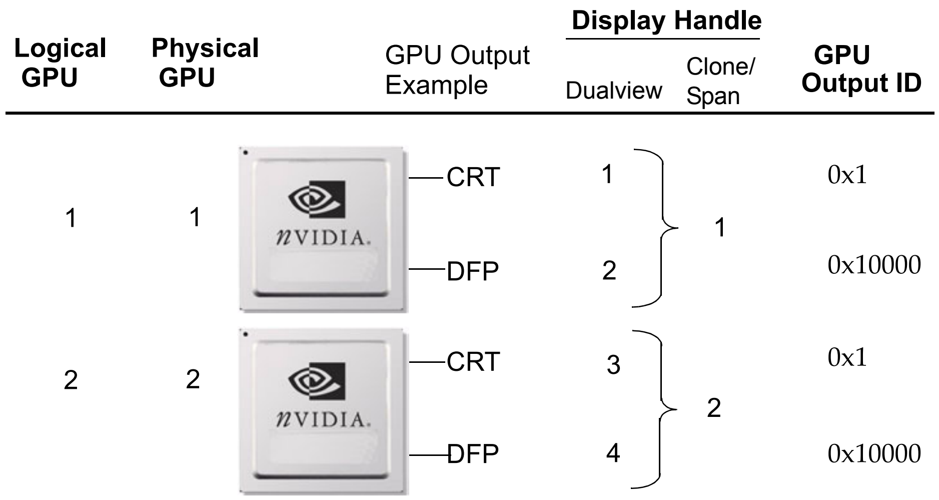

Figure 2.1 NvAPI Handles–Dualview, Clone and Spanning Modes

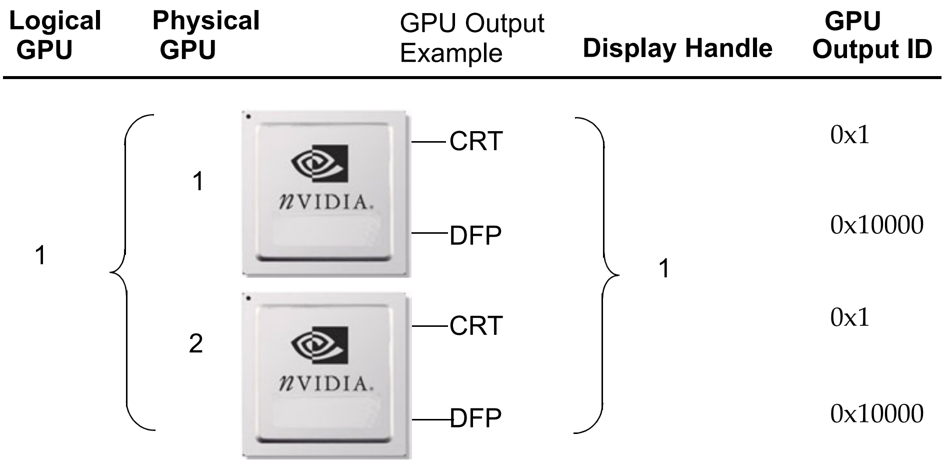

Figure 2.2 NvAPI Handles–SLI Mode

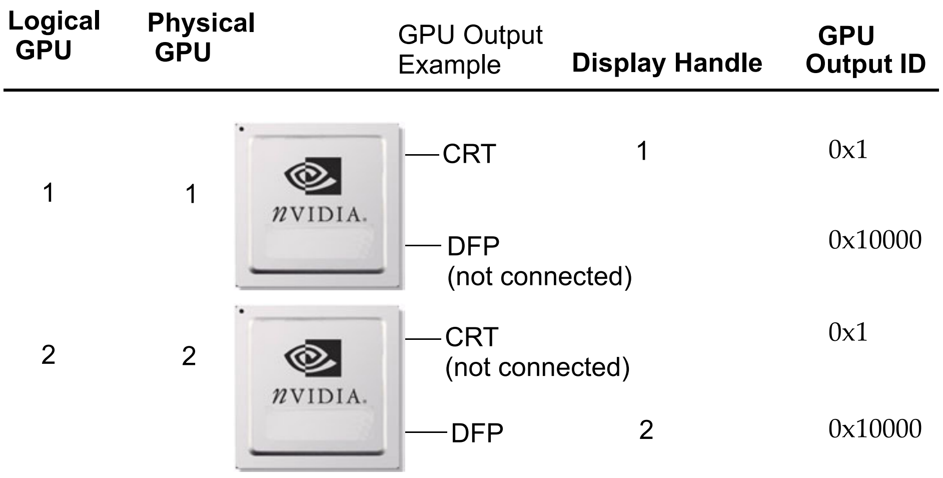

Figure 2.3 NvAPI Handles–Two GPUs Under Dualview

Structure Versions Must be Initialized

Each structure contains a version field which the caller must initialize so that the API library can track the version that is used by the calling application.

Each structure also has an associated NvAPI macro that you can use to initialize the version field. For example, the macro for structure **NV_XXX** is **NV_XXX_VER**. Initialize the version field as follows:

```
NV_XXX.version = NV_XXX_VER;
```

Use a Static Link with Applications

NvAPI cannot be dynamically linked to applications. You must create a static link to the library and then call NvAPI_Initialize(), which loads nvapi.dll dynamically.

If the NVIDIA drivers are not installed on the system or nvapi.dll is not present when the application calls NvAPI_Initialize(), the call just returns an error. The application will still load.

NvAPI Definitions

Index of NvAPI Calls

The following is an alphabetical listing of the API calls covered in this document.

Function	Earliest Driver Version	WinXP 32-bit	WinXp 64-bit	Vista 32-bit	Vista 64-bit
NvAPI_CreateDisplayFromUnattachedDisplay()	88.40	X	X	X	X
NvAPI_D3D9_AliasPrimaryAsTexture()	82.61	X	X	X	X
NvAPI_D3D9_AliasPrimaryFromDevice()	82.61	X	X	X	X
NvAPI_D3D9_GetSurfaceHandle()	82.61	X	X	X	X
NvAPI_D3D9_GetTextureHandle()	82.61	X	X	X	X
NvAPI_D3D9_GetCurrentRenderTargetHandle()	82.61	X	X	X	X
NvAPI_D3D9_GetCurrentZBufferHandle()	82.61	X	X	X	X
NvAPI_D3D9_GpuSyncAcquire()	92.00 95.40	X	X		
NvAPI_D3D9_GpuSyncEnd()	92.00 95.40	X	X		
NvAPI_D3D9_GpuSyncGetHandleSize()	92.00 95.40	X	X		
NvAPI_D3D9_GpuSyncInit()	92.00 95.40	X	X		
NvAPI_D3D9_GpuSyncMapTexBuffer()	92.00 95.40	X	X		
NvAPI_D3D9_GpuSyncMapVertexBuffer()	92.00 95.40	X	X		
NvAPI_D3D9_GpuSyncRelease()	92.00 95.40	X	X		
NvAPI_D3D9_Lock()	82.61	X	X	X	X
NvAPI_D3D9_LockForCUDA()	96.40	X	X		
NvAPI_D3D9_PresentSurfaceToDesktop()	82.61	X	X	X	X
NvAPI_D3D9_PresentVideo()	91.10 95.06	X	X	X	X
NvAPI_D3D9_RestoreDesktop()	82.61	X	X	X	X
NvAPI_D3D9_Unlock()	82.61	X	X	X	X

Function	Earliest Driver Version	WinXP 32-bit	WinXp 64-bit	Vista 32-bit	Vista 64-bit
NvAPI_DisableHWCursor()	82.61	X	X		
NvAPI_EnableHWCursor()	82.61	X	X		
NvAPI_EnumNvidiaDisplayHandle()	82.61	X	X	X	
NvAPI_EnumNvidiaUnAttachedDisplayHandle()	88.40	X	X	X	
NvAPI_EnumPhysicalGPUs()	82.61	X	X	X	X
NvAPI_EnumLogicalGPUs()	86.60	X	X	X	X
NvAPI_GetAssociatedDisplayOutputId())	96.80	X	X		
NvAPI_GetAssociatedNvidiaDisplayHandle()	82.61	X	X	X	X
NvAPI_GetAssociatedNvidiaDisplayName()	87.80	X	X	X	X
NvAPI_GetErrorMessage()	82.61	X	X	X	X
NvAPI_GetInterfaceVersionString()	82.61	X	X	X	X
NvAPI_GetDisplayDriverVersion()	82.61	X	X	X	X
NvAPI_GetHDMISupportInfo()	97.00	X	X		
NvAPI_GetInfoFrame()	97.00	X	X		
NvAPI_GetLogicalGPUFromDisplay()	86.60	X	X	X	X
NvAPI_GetLogicalGPUFromPhysicalGPU()	86.60	X	X	X	X
NvAPI_GetPhysicalGPUsFromDisplay()	86.60	X	X	X	X
NvAPI_GetPhysicalGPUsFromLogicalGPU()	86.60	X	X	X	X
NvAPI_GetPhysicalGPUFromUnattachedDisplay()	88.40	X	X	X	X
NvAPI_GetSupportedViews()	96.10			X	X
NvAPI_GetVBlankCounter()	90.19	X	X		
NvAPI_GetView()	96.10			X	X
NvAPI_GPU_GetAGPAperture()	92.10	X	X	X	X
NvAPI_GPU_GetAllOutputs()	87.00	X	X	X	X
NvAPI_GPU_GetActiveOutputs()	87.00	X	X	X	X
NvAPI_GPU_GetBusType()	92.10	X	X	X	X
NvAPI_GPU_GetConnectedOutputs()	87.00	X	X	X	X
NvAPI_GPU_GetConnectedOutputsWithLidState()	97.30	X	X	X	X
NvAPI_GPU_GetCurrentAGPRate()	92.10	X	X	X	X
NvAPI_GPU_GetCurrentPCIEDownstreamWidth()	92.10	X	X	X	X
NvAPI_GPU_GetEDID()	88.50	X	X	X	X
NvAPI_GPU_GetFullName()	92.10	X	X	X	X
NvAPI_GPU_GetIRQ()	92.10	X	X	X	X

Function	Earliest Driver Version	WinXP 32-bit	WinXp 64-bit	Vista 32-bit	Vista 64-bit
NvAPI_GPU_GetOutputType()	87.00	X	X	X	X
NvAPI_GPU_GetPCIIdentifiers()	92.10	X	X	X	X
NvAPI_GPU_GetPerfClocks()	92.40	X	X	X	X
NvAPI_GPU_GetPhysicalFrameBufferSize()	92.10	X	X	X	X
NvAPI_GPU_GetQuadroStatus()	96.30	X	X	X	X
NvAPI_GPU_GetThermalSettings())	92.40	X	X	X	X
NvAPI_GPU_GetVbiosRevision()	92.10	X	X	X	X
NvAPI_GPU_GetVbiosOEMRevision()	92.10	X	X	X	X
NvAPI_GPU_GetVbiosVersionString()	92.10	X	X	X	X
NvAPI_GPU_GetVirtualFrameBufferSize()	92.10	X	X	X	X
NvAPI_GPU_RestoreCoolerPolicyTable()	92.40	X	X	X	X
NvAPI_GPU_RestoreCoolerSettings()	92.40	X	X	X	X
NvAPI_GPU_SetCoolerLevels()	92.40	X	X	X	X
NvAPI_GPU_SetCoolerPolicyTable()	92.40	X	X	X	X
NvAPI_GPU_SetPerfClocks()	92.40	X	X	X	X
NvAPI_GPU_ValidateOutputCombination()	87.10	X	X	X	X
NvAPI_I2CRead()	87.90	X	X	X	X
NvAPI_I2CWrite()	87.90	X	X	X	X
NvAPI_Initialize()	82.61	X	X	X	X
NvAPI_OGL_ExpertModeGet() / NvAPI_OGL_ExpertModeSet()	84.11 88.00	X	X	X	X
NvAPI_OGL_ExpertModeDefaultsSet() / NvAPI_OGL_ExpertModeDefaultsGet()	84.11 88.00	X	X	X	X
NvAPI_SetInfoFrame()	97.00	X	X		
NvAPI_SetRefreshRateOverride()	87.30	X	X		
NvAPI_SetView()	96.10			X	X
NvAPI_SetViewEx()	96.30			X	X

Value Types

```
typedef unsigned __int64 NvU64;
typedef unsigned long    NvU32;
typedef long             NvS32;
typedef unsigned char    NvU8;
typedef char NvAPI_String[NVAPI_GENERIC_STRING_MAX];
typedef char NvAPI_LongString[NVAPI_LONG_STRING_MAX];
typedef char NvAPI_ShortString[NVAPI_SHORT_STRING_MAX];
```

Defaults and Limits

```
#define NVAPI_DEFAULT_HANDLE                  0
#define NVAPI_GENERIC_STRING_MAX           4096
#define NVAPI_LONG_STRING_MAX               256
#define NVAPI_SHORT_STRING_MAX               64
#define NVAPI_MAX_PHYSICAL_GPUS              64
#define NVAPI_MAX_LOGICAL_GPUS               64
#define NVAPI_MAX_AVAILABLE_GPU_TOPOLOGIES  256
#define NVAPI_MAX_GPU_TOPOLOGIES    NVAPI_MAX_PHYSICAL_GPUS
#define NVAPI_MAX_GPU_PER_TOPOLOGY            8
#define NVAPI_MAX_DISPLAY_HEADS               2
#define NVAPI_MAX_DISPLAYS          NVAPI_MAX_PHYSICAL_GPUS *
                                   NVAPI_MAX_DISPLAY_HEADS
```

NvAPI Return Status Codes

All functions return an **`NvAPI_Status`** value.

For example,

- Any function receiving an invalid argument will return `NVAPI_INVALID_ARGUMENT`.
- If communication with an NVIDIA display driver cannot be established, functions will return `NVAPI_NVIDIA_DEVICE_NOT_FOUND`.
- If one or more handles passed have been invalidated due to a modeset or SLI reconfiguration event, then `NVAPI_HANDLE_INVALIDATED` will be returned.

The following is a complete list of status codes.

`NVAPI_OK`	= 0	Success
`NVAPI_ERROR`	= -1	Generic error
`NVAPI_LIBRARY_NOT_FOUND`	= -2	NvAPI support library could not be loaded.
`NVAPI_NO_IMPLEMENTATION`	= -3	The function is not implemented in the current driver installation.
`NVAPI_API_NOT_INTIALIZED`	= -4	NvAPI_Initialize has not been called (successfully or otherwise).
`NVAPI_INVALID_ARGUMENT`	= -5	Invalid argument
`NVAPI_NVIDIA_DEVICE_NOT_FOUND`	= -6	No NVIDIA display driver was found.
`NVAPI_END_ENUMERATION`	= -7	No more items to enumerate.
`NVAPI_INVALID_HANDLE`	= -8	Invalid handle
`NVAPI_INCOMPATIBLE_STRUCT_VERSION`	= -9	An argument's structure version is not supported.
`NVAPI_HANDLE_INVALIDATED`	= -10	The handle is no longer valid (likely because of GPU or display re-configuration).
`NVAPI_OPENGL_CONTEXT_NOT_CURRENT`	= -11	No NVIDIA OpenGL context is current (but needs to be).
`NVAPI_EXPECTED_LOGICAL_GPU_HANDLE`	= -100	A logical GPU handle was expected for one or more parameters.
`NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE`	= -101	A physical GPU handle was expected for one or more parameters.
`NVAPI_EXPECTED_DISPLAY_HANDLE`	= -102	An NV display handle was expected for one or more parameters

`NVAPI_INVALID_COMBINATION`	`= -103,`	Used in some commands to indicate that the combination of parameters is not valid.
`NVAPI_NOT_SUPPORTED`	`= -104,`	The requested feature is not supported in the GPU.
`NVAPI_PORTID_NOT_FOUND`	`= -105,`	No port ID was found for the I2C transaction.
`NVAPI_EXPECTED_UNATTACHED_DISPLAY_HANDLE`	`= -106,`	Expected an unattached display handle as one of the input parameter.
`NVAPI_INVALID_PERF_LEVEL`	`= -107,`	Invalid performance level
`NVAPI_DEVICE_BUSY`	`= -108,`	Device is busy, request not fulfilled.
`NVAPI_NV_PERSIST_FILE_NOT_FOUND`	`= -109,`	NV persist file is not found.
`NVAPI_PERSIST_DATA_NOT_FOUND`	`= -110,`	NV persist data is not found.
`NVAPI_EXPECTED_TV_DISPLAY`	`= -111,`	Expected TV output display
`NVAPI_EXPECTED_TV_DISPLAY_ON_DCONNECTOR`	`= -112,`	Expected TV output on D Connector - HDTV_EIAJ4120.

NvAPI Function Descriptions

This section describes the NvAPI functions, organized in the following groups:

- “General Interface Calls” on page 16
- “Display Driver Calls” on page 18
- “Display Handle Calls” on page 19
- “GPU Discovery Calls” on page 24
- “GPU InformationCalls” on page 33
- “Display Control Calls” on page 50
- “DirectX Calls” on page 68
- “GPU Clock Control Calls” on page 89
- “OpenGL Related Calls” on page 94
- “I2C Calls” on page 98
- “GPU Cooler Calls” on page 101
- “Thermal API Calls” on page 111
- “GPU Topology Configuration Calls” on page 114

In the OS/architecture listing for each API, “32-bit” refers to the Intel x86 architecture and “64-bit” refers to the AMD 64-bit extensions to the x86 architecture.

General Interface Calls

This section describes the following API calls:

- NvAPI_Initialize()
- NvAPI_GetErrorMessage()
- NvAPI_GetInterfaceVersionString()

NvAPI_Initialize()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function initializes the NvAPI library. This must be called before calling other NvAPI_ functions.

Function Prototype

```
NvAPI_Status NvAPI_Initialize();
```

Return Status[i]

NVAPI_OK	Initialized
NVAPI_ERROR	An error occurred during the initialization process.
NVAPI_LIBRARYNOTFOUND	Failed to load NvAPI support library.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetErrorMessage()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function converts an NvAPI error code into a null-terminated string.

Function Prototype

```
NvAPI_Status NvAPI_GetErrorMessage
                    (NvAPI_Status      nr,
                     NvAPI_ShortString szDesc);
```

Input Parameter

nr	The error code to convert.

Output Parameter

szDesc	The string corresponding to the error code.

Return Status

See "NvAPI Return Status Codes" on page 13 for a list of possible return values.

NvAPI_GetInterfaceVersionString()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function returns a string describing the version of the NvAPI library. The contents of the string are human readable. Do not assume a fixed format.

Function Prototype

```
NvAPI_Status NvAPI_GetInterfaceVersionString
               (NvAPI_ShortString szDesc);
```

Output Parameter

szDesc	User readable string giving NvAPI version information

Return Status

See "NvAPI Return Status Codes" on page 13 for list of possible return values.

Display Driver Calls

This section describes the following API calls:

- NvAPI_GetDisplayDriverVersion()

NvAPI_GetDisplayDriverVersion()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

NV_DISPLAY_DRIVER_VERSION Structure: Version 1

This function returns a structure that describes aspects of the display driver build.

Use **NV_DISPLAY_DRIVER_VERSION_VER** to initialize the structure version.

Function Prototype

```
NvAPI_Status NvAPI_GetDisplayDriverVersion
               (NvDisplayHandle            hNvDisplay,
                NV_DISPLAY_DRIVER_VERSION *pVersion);
```

Input Parameter

hNvDisplay	NVIDIA display handle.

Output Parameter

pVersion	Pointer to the NV_DISPLAY_DRIVER_VERSION Structure.

Return Status[i]

```
NVAPI_OK

NVAPI_ERROR
```

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NV_DISPLAY_DRIVER_VERSION Structure

```
typedef struct
{
    NvU32                  version;        // Structure version
    NvU32                  drvVersion;
    NvU32                  bldChangeListNum;
    NvAPI_ShortString      szBuildBranchString;
    NvAPI_ShortString      szAdapterString;
} NV_DISPLAY_DRIVER_VERSION;
```

Display Handle Calls

This section describes the following API calls:

- NvAPI_CreateDisplayFromUnattachedDisplay()
- NvAPI_EnumNvidiaDisplayHandle()
- NvAPI_EnumNvidiaUnAttachedDisplayHandle()
- NvAPI_GetAssociatedNvidiaDisplayName()
- NvAPI_GetAssociatedNvidiaDisplayHandle()

NvAPI_CreateDisplayFromUnattachedDisplay()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 88.40

This function converts the unattached display handle to an active attached display handle. This puts the system into Dualview mode, with the driver automatically assigning the Dualview displays.

At least one GPU must be present in the system and running an NVIDIA display driver.

Function Prototype

```
NvAPI_Status NvAPI_GetPhysicalGPUFromUnattachedDisplay
            (NvUnattachedDisplayHandle    hNvUnattachedDisp,
             NvDisplayHandle             *pNvDisplay);
```

Input Parameter

hNvUnattachedDisp	The NVIDIA handle for the unattached display
pNvDisplay	Pointer to the created NVIDIA display handle

Return Status [i]

NVAPI_OK	One or more handles were returned
NVAPI_INVALID_ARGUMENT	**hNvDisp** is not valid; **nvGPUHandle** or **pGpuCount** is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_EnumNvidiaDisplayHandle()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit

Earliest ForceWare Version: 82.61

This function returns the handle of the NVIDIA display specified by the enum index **thisEnum**. The client should continue enumerating until the API returns NVAPI_END_ENUMERATION.

Note: Display handles can get invalidated on a modeset, so the calling applications need to re-enum the handles after every modeset.

Function Prototype

```
NvAPI_Status NvAPI_EnumNvidiaDisplayHandle
                (NvU32            thisEnum,
                 NvDisplayHandle  *pNvDispHandle);
```

Input Parameter

thisEnum	The index of the NVIDIA display.

Output Parameter

pNvDispHandle	Pointer to the NVIDIA display handle.

Return Status[i]

NVAPI_OK	Return a valid NvDisplayHandle based on the enum index.
NVAPI_INVALID_ARGUMENT	Either the handle pointer is NULL or the enum index is too big.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	There is no NVIDIA device found in the system.
NVAPI_END_ENUMERATION	There are no more display devices to enumerate.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

Sample Code

```
   NvAPI_Status        nvapiStatus;
   NvDisplayHandle     hDisplay_a[NVAPI_MAX_PHYSICAL_GPUS * 2] = {0};

   // Enumerate all display handles
    for(i=0,nvapiStatus=NVAPI_OK; nvapiStatus == NVAPI_OK; i++)
    {
     nvapiStatus = NvAPI_EnumNvidiaDisplayHandle(i, &hDisplay_a[i]);
```

Sample Code (continued)

```
      if (nvapiStatus == NVAPI_OK) nvDisplayCount++;
     }
     printf("     Displays: ");
     for(i=0; i<nvDisplayCount; i++)
     {
      Message(" %08x", hDisplay_a[i]);
     }
     printf("\n");
```

NvAPI_EnumNvidiaUnAttachedDisplayHandle()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit

Earliest ForceWare Version: 88.40

This function returns the handle of the unattached NVIDIA display specified by the enum index (**`thisEnum`**). The client should keep enumerating until an error is returned.

Note: Display handles can get invalidated on a modeset, so the calling applications need to re-enum the handles after every modeset.

Function Prototype

```
NvAPI_Status NvAPI_EnumNvidiaUnAttachedDisplayHandle
         (NvU32                          thisEnum,
          NvUnAttachedDisplayHandle   *pNvUnAttachedDispHandle);
```

Input Parameter

`thisEnum`	The index of the NVIDIA display.

Output Parameter

`pNvUnAttachedDispHandle`	Pointer to the NVIDIA display handle of the unattached display.

Return Status[i]

`NVAPI_OK`	Return a valid NvDisplayHandle based on the enum index.
`NVAPI_INVALID_ARGUMENT`	Either the handle pointer is NULL or the enum index is too big.
`NVAPI_NVIDIA_DEVICE_NOT_FOUND`	There is no NVIDIA device found in the system.
`NVAPI_END_ENUMERATION`	There are no more display devices to enumerate.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetAssociatedNvidiaDisplayHandle()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function returns the handle of the NVIDIA display that is associated with the display name given—for example "DISPLAY1".

Function Prototype

```
NvAPI_Status NvAPI_GetAssociatedNvidiaDisplayHandle
                (const char      *szDisplayName,
                 NvDisplayHandle *pNvDispHandle);
```

Input Parameter

szDisplayName	The display name

Output Parameter

pNvDispHandle	Pointer to the NVIDIA display handle

Return Status[i]

NVAPI_OK	***pNvDispHandle** is now valid.
NVAPI_INVALID_ARGUMENT	Either argument is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND:	No NVIDIA device maps to that display name.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetAssociatedNvidiaDisplayName()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.80

This function returns the name of the NVIDIA display– for example "DISPLAY1"–that is associated with the given display handle.

Function Prototype

```
NvAPI_Status NvAPI_GetAssociatedNvidiaDisplayName
               (NvDisplayHandle      NvDispHandle
                NvAPI_ShortString    szDisplayName);
```

Input Parameter

NvDispHandle	The NVIDIA display handle

Output Parameter

szDisplayName	The display name

Return Status[i]

NVAPI_OK	***pNvDispHandle** is now valid.
NVAPI_INVALID_ARGUMENT	Either argument is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND:	No NVIDIA device maps to that display name.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

GPU Discovery Calls

This section describes the following API calls:

- NvAPI_EnumPhysicalGPUs()
- NvAPI_EnumLogicalGPUs()
- NvAPI_GetPhysicalGPUsFromDisplay()
- NvAPI_GetPhysicalGPUFromUnattachedDisplay()
- NvAPI_GetLogicalGPUFromDisplay()
- NvAPI_GetLogicalGPUFromPhysicalGPU()
- NvAPI_GetPhysicalGPUsFromLogicalGPU()
- NvAPI_GPU_GetFullName()

NvAPI_EnumPhysicalGPUs()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function returns an array of physical GPU handles. Each handle represents a physical GPU present in the system. This includes GPUs that are part of an SLI configuration as well as GPUs that are not visible to the OS. The array **nvGPUHandle** will be filled with physical GPU handle values. The returned **gpuCount** determines how many entries in the array are valid.

At least one GPU must be present in the system and running an NVIDIA display driver.

Note: All GPU handles get invalidated on a modeset, so the calling applications need to re-enum the handles after every modeset.

Function Prototype

```
NvAPI_Status NvAPI_EnumPhysicalGPUs
    (NvPhysicalGpuHandle  nvGPUHandle[NVAPI_MAX_PHYSICAL_GPUS],
     NvU32                *pGpuCount);
```

Output Parameter

nvGPUHandle[]	The physical GPU handle
pGpuCount	Pointer to the number of actual GPU handle values

Return Status[i]

NVAPI_OK	One or more handles were returned

Return Status[i]

NVAPI_INVALID_ARGUMENT	**nvGPUHandle** or **pGpuCount** is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_EnumLogicalGPUs()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 86.60

This function returns an array of logical GPU handles. Each handle represents one or more GPUs acting in concert as a single graphics device. The array **nvGPUHandle** will be filled with logical GPU handle values. The returned **gpuCount** determines how many entries in the array are valid.

At least one GPU must be present in the system and running an NVIDIA display driver.

Note: All GPU handles get invalidated on a modeset, so the calling applications need to re-enum the handles after every modeset.

Function Prototype

```
NvAPI_Status NvAPI_EnumLogicalGPUs
     (NvLogicalGpuHandle  nvGPUHandle[NVAPI_MAX_LOGICAL_GPUS],
      NvU32               *pGpuCount);
```

Output Parameter

nvGPUHandle[]	The logical GPU handle
pGpuCount	Pointer to the number of actual GPU handle values

Return Status [i]

NVAPI_OK	One or more handles were returned.
NVAPI_INVALID_ARGUMENT	**nvGPUHandle** or **pGpuCount** is NULL
VAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetPhysicalGPUsFromDisplay()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 86.60

This function returns an array of physical GPU handles associated with the specified display. The array **`nvGPUHandle`** will be filled with physical GPU handle values. The returned **`gpuCount`** determines how many entries in the array are valid.

If the display corresponds to more than one physical GPU, the first GPU returned is the one with the attached active output.

At least one GPU must be present in the system and running an NVIDIA display driver.

Function Prototype

```
NvAPI_Status NvAPI_GetPhysicalGPUsFromDisplay
   (NvDisplayHandle      hNvDisp,
    NvPhysicalGpuHandle  nvGPUHandle[NVAPI_MAX_PHYSICAL_GPUS],
    NvU32               *pGpuCount);
```

Input Parameter

`hNvDisp`	The NVIDIA display handle

Output Parameter

`nvGPUHandle[]`	The physical GPU handle
`pGpuCount`	Pointer to the number of actual GPU handle values

Return Status [i]

`NVAPI_OK`	One or more handles were returned
`NVAPI_INVALID_ARGUMENT`	**`hNvDisp`** is not valid; **`nvGPUHandle`** or **`pGpuCount`** is NULL
`NVAPI_NVIDIA_DEVICE_NOT_FOUND`	No NVIDIA GPU driving a display was found.
`NVAPI_EXPECTED_LOGICAL_GPU_HANDLE`	**`hLogicalGPU`** was not a logical GPU handle.

i. See “NvAPI Return Status Codes” on page 13 for a list of other possible return codes.

NvAPI_GetPhysicalGPUFromUnattachedDisplay()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 88.40

This function returns the physical GPU handle associated with the specified unattached display.

At least one GPU must be present in the system and running an NVIDIA display driver.

Function Prototype

```
NvAPI_Status NvAPI_GetPhysicalGPUFromUnattachedDisplay
            (NvUnattachedDisplayHandle    hNvUnattachedDisp,
             NvPhysicalGpuHandle         *pPhysicalGpu);
```

Input Parameter

hNvUnattachedDisp	The NVIDIA handle for the unattached display

Output Parameter

pPhysicalGpu	Pointer to the physical GPU handle

Return Status [i]

NVAPI_OK	One or more handles were returned
NVAPI_INVALID_ARGUMENT	**hNvDisp** is not valid; **nvGPUHandle** or **pGpuCount** is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetLogicalGPUFromDisplay()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 86.60

This function returns the logical GPU handle associated with the specified display.

At least one GPU must be present in the system and running an NVIDIA display driver. .

Function Prototype

```
NvAPI_Status NvAPI_GetLogicalGPUFromDisplay
                    (NvDisplayHandle     hNvDisp,
                     NvLogicalGpuHandle *pLogicalGPU);
```

Input Parameter

hNvDisp	The NVIDIA display handle

Output Parameter

pLogicalGPU	Pointer to the logical GPU handle

Return Status [i]

NVAPI_OK	One or more handles were returned.
NVAPI_INVALID_ARGUMENT	**hNvDisp** is not valid; **pLogicalGPU** is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetLogicalGPUFromPhysicalGPU()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 86.60

This function returns the logical GPU handle associated with the specified physical GPU handle.

At least one GPU must be present in the system and running an NVIDIA display driver.

Function Prototype

```
NvAPI_Status NvAPI_GetLogicalGPUFromPhysicalGPU
                    (NvPhysicalGpuHandle   hPhysicalGPU,
                     NvLogicalGpuHandle   *pLogicalGPU);
```

Input Parameter

hPhysicalGPU	The physical GPU handle

Output Parameter

`pLogicalGPU`	Pointer to the logical GPU handle

Return Status [i]

`NVAPI_OK`	One or more handles were returned.
`NVAPI_INVALID_ARGUMENT`	**`hPhysicalGPU`** is not valid; **`pLogicalGPU`** is NULL
`NVAPI_NVIDIA_DEVICE_NOT_FOUND`	No NVIDIA GPU driving a display was found.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetPhysicalGPUsFromLogicalGPU()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 86.60

This function returns the physical GPU handles associated with the specified logical GPU handle. The array **`hPhysicalGPU`** will be filled with physical GPU handle values. The returned **`gpuCount`** determines how many entries in the array are valid.

At least one GPU must be present in the system and running an NVIDIA display driver.

Function Prototype

```
NvAPI_Status NvAPI_GetPhysicalGPUsFromLogicalGPU
   (NvLogicalGpuHandle   hLogicalGPU,
    NvPhysicalGpuHandle  hPhysicalGPU[NVAPI_MAX_PHYSICAL_GPUS],
    NvU32               *pGpuCount);
```

Input Parameter

`hLogicalGPU`	The logical GPU handle.

Output Parameter

`hPhysicalGPU[]`	The physical GPU handle
`pGpuCount`	Pointer to the number of actual GPU handle values

Return Status [i]

`NVAPI_OK`	One or more handles were returned.
`NVAPI_INVALID_ARGUMENT`	**`hLogicalGPU`** is not valid; **`hPhysicalGPU`** is NULL
`NVAPI_NVIDIA_DEVICE_NOT_FOUND`	No NVIDIA GPU driving a display was found.
`NVAPI_EXPECTED_LOGICAL_GPU_HANDLE`	**hLogicalGPU** was not a logical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetFullName()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function retrieves the full GPU name as an ASCII string–for example, "Quadro FX 1400". .

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetFullName(
                NvDisplayHandle   hNvDisplay,
                NvAPI_ShortString szName);
```

Input Parameter

hNvDisplay	Handle for the display

Output Parameter

szName	GPU name

Return Status[i]

NVAPI_OK
NVAPI_ERROR

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetQuadroStatus()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.80

This function indicates whether the GPU is in the Quadro or GeForce family.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetQuadroStatus(
                NvDisplayHandle  hNvDisplay,
                NvU32           *pStatus);
```

Input Parameter

hNvDisplay	Handle for the display.

Output Parameter

pStatus	Pointer to the Quadro status. 1 = Quadro. 0 = GeForce.

Return Status[i]

NVAPI_OK
NVAPI_ERROR

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

GPU InformationCalls

This section describes the following API calls:

- NvAPI_GPU_GetAllOutputs()
- NvAPI_GPU_GetConnectedOutputs()
- NvAPI_GPU_GetActiveOutputs()
- NvAPI_GPU_GetEDID()
- NvAPI_GPU_GetOutputType()
- NvAPI_GPU_ValidateOutputCombination()
- NvAPI_GPU_GetPCIIdentifiers()
- NvAPI_GPU_GetBusType()
- NvAPI_GPU_GetIRQ()
- NvAPI_GPU_GetVbiosRevision()
- NvAPI_GPU_GetVbiosOEMRevision()
- NvAPI_GPU_GetVbiosVersionString()
- NvAPI_GPU_GetAGPAperture()
- NvAPI_GPU_GetCurrentAGPRate()
- NvAPI_GPU_GetCurrentPCIEDownstreamWidth()
- NvAPI_GPU_GetPhysicalFrameBufferSize()
- NvAPI_GPU_GetVirtualFrameBufferSize()

See "NvAPI Handles" on page 6 for an explanation of GPU output identifiers.

GPU Structures and Enums

NV_GPU_CONNECTOR_TYPE

```
typedef enum _NV_GPU_CONNECTOR_TYPE
{
    NVAPI_GPU_CONNECTOR_VGA_15_PIN              = 0x00000000,
    NVAPI_GPU_CONNECTOR_TV_COMPOSITE            = 0x00000010,
    NVAPI_GPU_CONNECTOR_TV_SVIDEO               = 0x00000011,
    NVAPI_GPU_CONNECTOR_TV_HDTV_COMPONENT       = 0x00000013,
    NVAPI_GPU_CONNECTOR_TV_SCART                = 0x00000014,
    NVAPI_GPU_CONNECTOR_TV_HDTV_EIAJ4120        = 0x00000017,
    NVAPI_GPU_CONNECTOR_PC_POD_HDTV_YPRPB       = 0x00000018,
    NVAPI_GPU_CONNECTOR_PC_POD_SVIDEO           = 0x00000019,
    NVAPI_GPU_CONNECTOR_PC_POD_COMPOSITE        = 0x0000001A,
    NVAPI_GPU_CONNECTOR_DVI_I_TV_SVIDEO         = 0x00000020,
    NVAPI_GPU_CONNECTOR_DVI_I_TV_COMPOSITE      = 0x00000021,
    NVAPI_GPU_CONNECTOR_DVI_I                   = 0x00000030,
    NVAPI_GPU_CONNECTOR_DVI_D                   = 0x00000031,
    NVAPI_GPU_CONNECTOR_ADC                     = 0x00000032,
    NVAPI_GPU_CONNECTOR_LFH_DVI_I_1             = 0x00000038,
    NVAPI_GPU_CONNECTOR_LFH_DVI_I_2             = 0x00000039,
    NVAPI_GPU_CONNECTOR_SPWG                    = 0x00000040,
    NVAPI_GPU_CONNECTOR_OEM                     = 0x00000041,
    NVAPI_GPU_CONNECTOR_HDMI_A                  = 0x00000061,
    NVAPI_GPU_CONNECTOR_UNKNOWN                 = 0xFFFFFFFF,
} NV_GPU_CONNECTOR_TYPE;
```

NvAPI_GPU_GetAllOutputs()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.00

For the specified GPU, this function returns a set of all GPU-output identifiers as a bitmask.

Function Prototype

```
NvAPI_Status NvAPI_GPU_GetAllOutputs
               (NvPhysicalGpuHandle  hPhysicalGpu,
                NvU32                *pOutputsMask);
```

Input Parameter

hPhysicalGpu	The physical GPU handle

Output Parameter

pOutputsMask	Pointer to the bit mask indicating the GPU output identifiers.

Return Status [i]

NVAPI_OK	***pOutputsMask** contains a set of GPU-output identifiers.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or **pOutputsMask** is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See “NvAPI Return Status Codes” on page 13 for a list of other possible return codes.

NvAPI_GPU_GetConnectedOutputs()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.00

This function is the same as NvAPI_GPU_GetAllOutputs() but returns only the set of GPU-output identifiers that are connected to display devices.

Function Prototype

```
NvAPI_Status NvAPI_GPU_GetConnectedOutputs
               (NvPhysicalGpuHandle  hPhysicalGpu,
                NvU32                *pOutputsMask);
```

Input Parameter

hPhysicalGpu	The physical GPU handle

Output Parameter

pOutputsMask	Pointer to the bit mask indicating the GPU output identifiers.

Return Status [i]

NVAPI_OK	***pOutputsMask** contains a set of GPU-output identifiers.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or **pOutputsMask** is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetConnectedOutputsWithLidState()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 97.30

This function is the same as NvAPI_GPU_GetConnectedOutputs() but returns the connected display identifiers that are connected as an output mask, but unlike NvAPI_GPU_GetConnectedOutputs, this API "always" reflects the Lid State in the output mask.

Thus, use this API if you expect the LID close state to be available in the connection mask.

- If LID is closed, then this API will remove the LID panel from the connected display identifiers.
- If LID is open, then this API will reflect the LID panel in the connected display identifiers.

This API should be used on laptop systems and on systems where the LID state is required in the connection output mask.

On desktop systems, the returned identifiers will match NvAPI_GPU_GetConnectedOutputs.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetConnectedOutputsWithLidState(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32              *pOutputsMask);
```

Input Parameter

hPhysicalGpu	The physical GPU handle

Output Parameter

pOutputsMask	Pointer to the bit mask indicating the GPU output identifiers.

Return Status [i]

NVAPI_OK	***pOutputsMask** contains a set of GPU-output identifiers.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or **pOutputsMask** is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetActiveOutputs()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.00

This function is the same as NvAPI_GPU_GetAllOutputs() but returns only the set of GPU-output identifiers that are actively driving display devices.

Function Prototype

```
NvAPI_Status NvAPI_GPU_GetActiveOutputs
               (NvPhysicalGpuHandle  hPhysicalGpu,
                NvU32                *pOutputsMask);
```

Input Parameter

hPhysicalGpu	The physical GPU handle

Output Parameter

pOutputsMask	Pointer to the bit mask indicating the GPU output identifiers.

Return Status [i]

NVAPI_OK	***pOutputsMask** contains a set of GPU-output identifiers.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or **pOutputsMask** is NULL
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See “NvAPI Return Status Codes” on page 13 for a list of other possible return codes.

NvAPI_GPU_GetEDID()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 88.50

NV_EDID Structure: Version 1

This function returns the EDID data for the specified GPU handle and connection bit mask. **`outputsMask`** should have only one bit set in order to indicate a single display.

Use **`NV_EDID_VER`** to initialize the structure version.

Function Prototype

```
NVAPI_Status NvAPI_GPU_GetEDID(
             NvPhysicalGpuHandle hPhysicalGpu,
             NvU32               outputsMask,
             NV_EDID            *pEDID);
```

Input Parameter

`hPhysicalGpu`	The physical GPU handle
`outputsMask`	To the bit mask indicating the GPU output identifiers.

Output Parameter

`pEDID`	Pointer to the EDID data (NV_EDID structure).

Return Status [i]

NVAPI_OK	**`pEDID`** contains valid data
NVAPI_INVALID_ARGUMENT	**`pEDID`** is NULL, or **`outputsMask`** has 0 or > 1 bits set.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**`hPhysicalGpu`** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NV_EDID Structure

```
typedef struct
{
    NvU32   version;        //structure version
    NvU8    EDID_Data[NV_EDID_DATA_SIZE];
} NV_EDID;
```

```
#define NV_EDID_DATA_SIZE   256
```

NvAPI_GPU_GetOutputType()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.00

This function determines the display type (CRT, DFP, or TV) corresponding to a particular physical GPU handle and output ID.

Function Prototype

```
NvAPI_Status NvAPI_GPU_GetOutputType
               (NvPhysicalGpuHandle  hPhysicalGpu,
                NvU32                outputId,
                NV_GPU_OUTPUT_TYPE  *pOutputType);
```

Input Parameter

hPhysicalGpu	The physical GPU handle
outputId	The output ID of the specified GPU

Output Parameter

pOutputType	One of the display types enumerated in NV_GPU_OUTPUT_TYPE.

Return Status [i]

NVAPI_OK	***pOutputType** contains a NvGpuOutputType value.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu**, **outputId** or **pOutputsMask** is NULL; or **outputId** has > 1 bit set.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NV_GPU_OUTPUT_TYPE

```
typedef enum _NV_GPU_OUTPUT_TYPE
 {
   NVAPI_GPU_OUTPUT_UNKNOWN  = 0,
   NVAPI_GPU_OUTPUT_CRT      = 1,    // CRT display device
   NVAPI_GPU_OUTPUT_DFP      = 2,    // Digital Flat Panel
                                        display device
   NVAPI_GPU_OUTPUT_TV       = 3,    // TV display device
 } NV_GPU_OUTPUT_TYPE;
```

NvAPI_GPU_ValidateOutputCombination()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.10

This function determines if a set of GPU outputs can be active simultaneously. Typically, on GPUs with more than one output, all the outputs cannot be active at the same time due to internal resource sharing.

Use NvAPI_GPU_GetAllOutputs() to determine which outputs are candidates.

Function Prototype

```
NvAPI_Status NvAPI_GPU_ValidateOutputCombination(
              NvPhysicalGpuHandle hPhysicalGpu,
              NvU32               outputsMask);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
outputsMask	The set of GPU-output identifiers.

Return Status [i]

NVAPI_OK	The combination of outputs in **outputsMask** can be active simultaneously.
NVAPI_INVALID_COMBINATION	The combination of outputs in **outputsMask** are NOT valid
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or **OutputsMask** does not have at least two bits set.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See “NvAPI Return Status Codes” on page 13 for a list of other possible return codes.

Return Status

NVAPI_INVALID_ARGUMENT	**hPhysicalGpu**, **outputId** or **pOutputsMask** is NULL; or outputId has > 1 bit set
NVAPI_OK	***pConnectorInfo** contains valid NV_GPU_CONNECTOR_INFO data
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle
NVAPI_INCOMPATIBLE_STRUCT_VERSION	**NV_GPU_CONNECTOR_INFO** version not compatible with driver

NvAPI_GPU_GetPCIIdentifiers()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the PCI identifiers associated with the specified GPU.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetPCIIdentifiers(
      NvPhysicalGpuHandle  hPhysicalGpu,
      NvU32               *pDeviceId,
      NvU32               *pSubSystemId,
      NvU32               *pRevisionId,
      NvU32               *pExtDeviceId);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pDeviceId	The internal PCI device identifier for the GPU
pSubSystemId	The internal PCI subsystem identifier for the GPU.
pRevisionId	The internal PCI device-specific revision identifier for the GPU
pExtDeviceId	The external PCI device identifier for the GPU.

Return Status [i]

NVAPI_OK	Arguments are populated with PCI identifiers
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.

Return Status [i]

NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetBusType()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the type of bus associated the specified GPU.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetBusType(
                NvPhysicalGpuHandle  hPhysicalGpu,
                NV_GPU_BUS_TYPE      *pBusType);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pBusTyped	Pointer to the bus type. See NV_GPU_BUS_TYPE.

Return Status [i]

NVAPI_OK	pBusType contians the bus identifier.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NV_GPU_BUS_TYPE

```
typedef enum _NV_GPU_BUS_TYPE
{
    NVAPI_GPU_BUS_TYPE_UNDEFINED    = 0,
    NVAPI_GPU_BUS_TYPE_PCI          = 1,
    NVAPI_GPU_BUS_TYPE_AGP          = 2,
    NVAPI_GPU_BUS_TYPE_PCI_EXPRESS  = 3,
```

NV_GPU_BUS_TYPE

```
    NVAPI_GPU_BUS_TYPE_FPCI          = 4,
} NV_GPU_BUS_TYPE;
```

NvAPI_GPU_GetIRQ()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function rReturns the interrupt number associated the specified GPU.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetIRQ(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32               *pIRQ);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pIRQ	Pointer to the interrupt number.

Return Status [i]

NVAPI_OK	pIRQ contains the interrupt number.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or pIRQ is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetVbiosRevision()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the revision of the video BIOS associated the specified GPU.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetVbiosRevision(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32               *pBiosRevision);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pBiosRevision	Pointer to the BIOS revision

Return Status [i]

NVAPI_OK	**pBiosRevision** contains the revision number.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See “NvAPI Return Status Codes” on page 13 for a list of other possible return codes.

NvAPI_GPU_GetVbiosOEMRevision()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the OEM revision of the video BIOS associated the specified GPU.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetVbiosOEMRevision(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32               *pBiosRevision);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pBiosRevision	Pointer to the BIOS revision

Return Status [i]

NVAPI_OK	**pBiosRevision** contains the revision number.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetVbiosVersionString()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the full BIOS version string in the form of xx.xx.xx.xx.yy where

- The xx numbers come from NvAPI_GPU_GetVbiosRevision, and
- yy comes from NvAPI_GPU_GetVbiosOEMRevision.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetVbiosVersionString(
                NvPhysicalGpuHandle  hPhysicalGpu,
                NvAPI_ShortString    szBiosRevision);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
szBiosRevision	The full revision string

Return Status [i]

NVAPI_OK	**szBiosRevision** contains the version string.
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetAGPAperture()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the AGP aperture in megabytes (MB).

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetAGPAperture(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32              *pSize);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pSize	Pointer to the AGP aperture size

Return Status [i]

NVAPI_OK	Success
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetCurrentAGPRate()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the current AGP Rate (1 = 1x, 2=2x etc, 0 = AGP not present).

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetCurrentAGPRate(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32               *pRate);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pRate	Pointer to the current AGP rate. (0 = AGP not present, 1 = 1x, 2=2x, etc.)

Return Status [i]

NVAPI_OK	Success
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetCurrentPCIEDownstreamWidth()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the number of PCI Express lanes being used for the PCIE interface downstream from the GPU.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetCurrentPCIEDownstreamWidth(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32               *pWidth);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pWidth	Pointer to the PCIE lane width.

Return Status [i]

NVAPI_OK	Success
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetPhysicalFrameBufferSize()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the physical size of the frame buffer in kilobytes (KB). This does NOT include any system RAM that may be dedicated for use by the GPU.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetPhysicalFrameBufferSize(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32               *pSize);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pSize	The frame buffer size (KB).

Return Status [i]

NVAPI_OK	Success
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetVirtualFrameBufferSize()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 92.10

This function returns the virtual size of the frame buffer in kilobytes (KB). This includes the physical RAM plus any system RAM that has been dedicated for use by the GPU.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetVirtualFrameBufferSize(
                NvPhysicalGpuHandle hPhysicalGpu,
                NvU32               *pSize);
```

Input Parameters

hPhysicalGpu	The physical GPU handle
pSize	Pointer to the size of the virtual frame buffer (KB)

Return Status [i]

NVAPI_OK	Success
NVAPI_INVALID_ARGUMENT	**hPhysicalGpu** or an argument is NULL.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVidia GPU driving a display was found
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	**hPhysicalGpu** was not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

Display Control Calls

This section describes the following API calls:

- NvAPI_EnableHWCursor()
- NvAPI_DisableHWCursor()
- NvAPI_GetVBlankCounter()
- NvAPI_SetView()
- NvAPI_GetView()
- NvAPI_SetViewEx()
- NvAPI_GetSupportedViews()
- NvAPI_SetRefreshRateOverride()
- NvAPI_GetAssociatedDisplayOutputId())
- NvAPI_GetHDMISupportInfo()
- NvAPI_GetInfoFrame()
- NvAPI_SetInfoFrame()

Display Control Structures and Enums

```
#define NVAPI_MAX_VIEW_TARGET  2
```

NV_VIEW_TARGET_INFO Struct

```
typedef struct
{
    NvU32 version;     // (IN) structure version
    NvU32 count;       // (IN) target count
    struct
    {
        NvU32 deviceMask;  // (IN/OUT) device mask
        NvU32 sourceId;    // (IN/OUT) source id
        NvU32 bPrimary:1;  // (OUT) Indicates if this is the
                              desktop primary
    } target[NVAPI_MAX_VIEW_TARGET];
} NV_VIEW_TARGET_INFO;
```

NV_TARGET_VIEW_MODE

```
typedef enum _NV_TARGET_VIEW_MODE
{
```

NV_TARGET_VIEW_MODE

```
    NV_VIEW_MODE_STANDARD  = 0,
    NV_VIEW_MODE_CLONE     = 1,
    NV_VIEW_MODE_HSPAN     = 2,
    NV_VIEW_MODE_VSPAN     = 3,
    NV_VIEW_MODE_DUALVIEW  = 4,
    NV_VIEW_MODE_MULTIVIEW = 5,
} NV_TARGET_VIEW_MODE;
```

Scaling Modes

NV_SCALING

```
typedef enum _NV_SCALING
{
    NV_SCALING_DEFAULT          = 0,        // No change
    NV_SCALING_MONITOR_SCALING  = 1,
    NV_SCALING_ADAPTER_SCALING  = 2,
    NV_SCALING_CENTERED         = 3,
    NV_SCALING_ASPECT_SCALING   = 5,
    NV_SCALING_CUSTOMIZED       = 255       // For future use
} NV_SCALING;
```

Rotate Modes

NV_ROTATE

```
typedef enum _NV_ROTATE
{
    NV_ROTATE_0           = 0,
    NV_ROTATE_90          = 1,
    NV_ROTATE_180         = 2,
    NV_ROTATE_270         = 3
} NV_ROTATE;
```

Color Formats

NV_FORMAT

```
typedef enum _NV_FORMAT
{
    NV_FORMAT_UNKNOWN       =  0,       // unknown. Driver will
choose one as following value.
    NV_FORMAT_P8              = 41,       // for 8bpp mode
    NV_FORMAT_R5G6B5          = 23,       // for 16bpp mode
    NV_FORMAT_A8R8G8B8        = 21,       // for 32bpp mode
    NV_FORMAT_A16B16G16R16F = 113       // for 64bpp(floating
point) mode.
```

NV_FORMAT

```
} NV_FORMAT;
```

NV_DISPLAY_TV_FORMAT

```
typedef enum _NV_DISPLAY_TV_FORMAT
{
    NV_DISPLAY_TV_FORMAT_NONE           = 0,
    NV_DISPLAY_TV_FORMAT_SD_NTSCM       = 0x00000001,
    NV_DISPLAY_TV_FORMAT_SD_NTSCJ       = 0x00000002,
    NV_DISPLAY_TV_FORMAT_SD_PALM        = 0x00000004,
    NV_DISPLAY_TV_FORMAT_SD_PALBDGH     = 0x00000008,
    NV_DISPLAY_TV_FORMAT_SD_PALN        = 0x00000010,
    NV_DISPLAY_TV_FORMAT_SD_PALNC       = 0x00000020,
    NV_DISPLAY_TV_FORMAT_SD_576i        = 0x00000100,
    NV_DISPLAY_TV_FORMAT_SD_480i        = 0x00000200,
    NV_DISPLAY_TV_FORMAT_ED_480p        = 0x00000400,
    NV_DISPLAY_TV_FORMAT_ED_576p        = 0x00000800,
    NV_DISPLAY_TV_FORMAT_HD_720p        = 0x00001000,
    NV_DISPLAY_TV_FORMAT_HD_1080i       = 0x00002000,
    NV_DISPLAY_TV_FORMAT_HD_1080p       = 0x00004000,
    NV_DISPLAY_TV_FORMAT_HD_720p50      = 0x00008000,
    NV_DISPLAY_TV_FORMAT_HD_1080p24     = 0x00010000,
    NV_DISPLAY_TV_FORMAT_HD_1080i50     = 0x00020000,

} NV_DISPLAY_TV_FORMAT;
```

NvAPI_EnableHWCursor()

OS/architecture: Windows XP/32-bit and 64-bit,
Earliest ForceWare Version: 82.61

This function enables hardware cursor support for the display specified by the NVIDIA display handle.

Note: Under Clone or Spanning mode, the display handle is associated with two monitors and the call will affect both monitors.

Function Prototype

```
NvAPI_Status NvAPI_EnableHWCursor
               (NvDisplayHandle hNvDisplay);
```

Input Parameter

hNvDisplay	The handle of the display for which to enable cursor support.

Return Status [i]

```
NVAPI_OK
NVAPI_ERROR
```

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_DisableHWCursor()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 82.61

This function enables hardware cursor support for the display specified by the NVIDIA display handle.

Note: Under Clone or Spanning mode, the display handle is associated with two monitors and the call will affect both monitors.

Function Prototype

```
NvAPI_Status NvAPI_DisableHWCursor
               (NvDisplayHandle hNvDisplay);
```

Input Parameter

hNvDisplay	The handle of the display for which to disable cursor support.

Return Status [i]

```
NVAPI_OK
NVAPI_ERROR
```

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetVBlankCounter()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 90.19 (Rel90)

This function gets the location of the VBlank counter.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GetVBlankCounter(
                NvDisplayHandle  hNvDisplay,
                NvU32           *pCounter);
```

Input Parameter

hNvDisplay	The handle of the display for which to get the VBlank count.

Output Parameter

pCounter	Pointer to the VBlank counter.

Return Status [i]

NVAPI_OK
NVAPI_ERROR

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_SetView()

OS/architecture : Windows Vista / 32-bit and 64-bit
Earliest ForceWare Version:
NV_VIEW_TARGET_INFO Structure: Version 1

This function modifies the target display arrangement for the selected display handle to any nView mode. It can also modify or extend the source display in Dualview mode.

Use **NV_VIEW_TARGET_INFO_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_SetView(
                NvDisplayHandle      hNvDisplay,
                NV_VIEW_TARGET_INFO *pTargetInfo,
                NV_TARGET_VIEW_MODE  targetView);
```

Input Parameter

hNvDisplay	NVIDIA Display selection. It can be **NVAPI_DEFAULT_HANDLE** or a handle enumerated from **NvAPI_EnumNVidiaDisplayHandle**().
pTargetInfo	Pointer to array of NV_VIEW_TARGET_INFO, specifying device properties in this view. The first device entry in the array is the physical primary. The device entry with the lowest source ID is the desktop primary. See "NV_VIEW_TARGET_INFO Struct" on page 50.
targetView	Target view selected from NV_TARGET_VIEW_MODE.

Return Status[i]

NVAPI_OK	Request completed.
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_INVALID_ARGUMENT	Invalid input parameter

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetView()

OS/architecture : Windows Vista / 32-bit and 64-bit
Earliest ForceWare Version:
NVAPI_VIEW_TARGET_INFO Structure: Version 1

This function retrieves the target display arrangement for the selected source display handle.

Use **NVAPI_VIEW_TARGET_INFO_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GetView(
                NvDisplayHandle       hNvDisplay,
                NV_VIEW_TARGET_INFO *pTargets,
                NvU32               *pTargetMaskCount,
                NV_TARGET_VIEW_MODE *pTargetView);
```

Input Parameter

hNvDisplay	NVIDIA Display selection. It can be **NVAPI_DEFAULT_HANDLE** or a handle enumerated from **NvAPI_EnumNVidiaDisplayHandle**().
targetMaskCount	Count of target device mask specified in pTargets.

Output Parameter

pTargets	User allocated storage to retrieve an array of NV_VIEW_TARGET_INFO. Can be NULL to retrieve the targetCount. See "NV_VIEW_TARGET_INFO Struct" on page 50.
ptargetMaskCount	Count of target device mask specified in pTargets.
ptargetView	Target view selected from NV_TARGET_VIEW_MODE.

Return Status[i]

NVAPI_OK	Request completed.
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_INVALID_ARGUMENT	Invalid input parameter

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_SetViewEx()

OS/architecture : Windows Vista / 32-bit and 64-bit
Earliest ForceWare Version: 97.30
NV_DISPLAY_PATH_INFO Structure: Version 1

This function modifies the target display arrangement for the selected display handle to any nView mode. It can also modify or extend the source display in Dualview mode.

Use **NV_DISPLAY_PATH_INFO_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_SetViewEx(
                NvDisplayHandle      hNvDisplay,
                NV_DISPLAY_PATH_INFO *pPathInfo,
                NV_TARGET_VIEW_MODE  displayView);
```

Input Parameter

hNvDisplay	NVIDIA Display selection. It can be **NVAPI_DEFAULT_HANDLE** or a handle enumerated from **NvAPI_EnumNVidiaDisplayHandle**().
pPathInfo	Pointer to array of NV_DISPLAY_PATH_INFO, specifying device properties in this view. The first device entry in the array is the physical primary. The device entry with the lowest source ID is the desktop primary. See NV_DISPLAY_PATH_INFO.
displayView	Target view selected from NV_TARGET_VIEW_MODE.

Return Status[i]

NVAPI_OK	Request completed.
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_INVALID_ARGUMENT	Invalid input parameter

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NV_DISPLAY_PATH_INFO

```
typedef struct
{
    NvU32 version;      // (IN) structure version
    NvU32 count;        // (IN) path count
    struct
    {
        NvU32             deviceMask;    // (IN) device mask
```

NV_DISPLAY_PATH_INFO

```
        NvU32              sourceId;    // (IN) source id
        NvU32              bPrimary:1;  // (OUT)Indicates if this
                                           is the desktop primary
        NV_GPU_CONNECTOR_TYPE   connector;    // (IN) Specify
                                     connector type. For TV only.

        // source mode information
        NvU32              width;       // (IN) width of the mode
        NvU32              height;      // (IN) height of the mode
        NvU32              depth;       // (IN) depth of the mode
        NV_FORMAT          colorFormat; // Color format if it needs
                                          to be specified.
                                          Not currently used.

        //rotation setting of the mode
        NV_ROTATE          rotation;    // (IN) rotation setting.

        // the scaling mode
        NV_SCALING          scaling;    // (IN) scaling setting

        // Timing info
        NvU32              refreshRate; // (IN) refresh rate of the
                                           mode
        NvU32               interlaced:1; // (IN) interlaced mode
                                           flag

        NV_DISPLAY_TV_FORMAT    tvFormat;    // (IN) Only make
sence for HDTV. For SDTV, we should follow the setting.
    } path[NVAPI_MAX_DISPLAY_PATH];
} NV_DISPLAY_PATH_INFO;
```

NvAPI_GetSupportedViews()

OS/architecture : Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version:

This function enumerates all the supported NVIDIA display views—nView and Dualview modes.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GetSupportedViews(
                NvDisplayHandle      hNvDisplay,
                NV_TARGET_VIEW_MODE *pTargetViews,
                NvU32               *pViewCount);
```

Input Parameter

`hNvDisplay`	NVIDIA Display selection. It can be **`NVAPI_DEFAULT_HANDLE`** or a handle enumerated from **`NvAPI_EnumNVidiaDisplayHandle`**().
`pViewCount`	Count of supported views.

Output Parameter

`pTargetViews`	Array of supported views. Can be NULL to retrieve the pViewCount first. See "NV_TARGET_VIEW_MODE" on page 50.
`pViewCount`	Count of supported views.

Return Status[i]

`NVAPI_OK`	Request completed.
`NVAPI_ERROR`	Miscellaneous error occurred.
`NVAPI_INVALID_ARGUMENT`	Invalid input parameter

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_SetRefreshRateOverride()

OS/architecture : Windows XP / 32-bit and 64-bit
Earliest ForceWare Version: 87.30

This function overrides the refresh rate on the given display/outputsMask.

The new refresh rate can either be applied right away or deferred to occur at the next OS modeset. The override occurs on a one-time basis.

```
NVAPI_INTERFACE NvAPI_SetRefreshRateOverride(
                NvDisplayHandle hNvDisplay,
                NvU32 outputsMask,
                float refreshRate,
                NvU32 bSetDeferred);
```

Input Parameter

hNvDisplay	The NVIDIA display handle. It can be NVAPI_DEFAULT_HANDLE or a handle enumerated from NvAPI_EnumNVidiaDisplayHandle().
outputsMask	A set of bits that identify all target outputs that are associated with the NVIDIA display handle. When SLI is enabled, **outputsMask** applies only to the GPU that is driving the display output.
refreshRate	The refresh rate to set. "0.0" means cancel the override.
bSetDeferred	1 = Defer the refresh rate change for the next OS mode set. 0 = Change the refresh rate immediately.

Return Status[i]

NVAPI_OK	The refresh rate override is correctly set.
NVAPI_ERROR	The operation failed
NVAPI_INVALID_ARGUMENT	**hNvDisplay** or **outputsMask** is invalid.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetAssociatedDisplayOutputId())

OS/architecture : Windows XP / 32-bit and 64-bit
Earliest ForceWare Version: 96.80

This function gets the active outputId associated with the display handle.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GetAssociatedDisplayOutputId(
                NvDisplayHandle hNvDisplay,
                NvU32           *pOutputId);
```

Input Parameter

hNvDisplay	The NVIDIA display handle. It can be NVAPI_DEFAULT_HANDLE or a handle enumerated from NvAPI_EnumNVidiaDisplayHandle().

Output Parameter

outputId	The active display output ID associated with the selected display handle hNvDisplay. The outputid will have only one bit set. In the Clone or Span case, this indicates the display outputId of the primay display that the GPU is driving.

Return Status[i]

NVAPI_OK	Success
NVAPI_ERROR	The operation failed
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_EXPECTED_DISPLAY_HANDLE	hNvDisplay is not a valid display handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GetHDMISupportInfo()

OS/architecture : Windows XP / 32-bit and 64-bit
Earliest ForceWare Version: 97.00
NV_HDMI_SUPPORT_INFO: Version 1

This API returns the current infoframe data on the specified display.

Use **NV_HDMI_SUPPORT_INFO_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GetHDMISupportInfo(
                NvDisplayHandle      hNvDisplay,
                NvU32                outputId,
                NV_HDMI_SUPPORT_INFO *pInfo);
```

Input Parameter

hNvDisplay	The NVIDIA display handle. It can be NVAPI_DEFAULT_HANDLE or a handle enumerated from NvAPI_EnumNVidiaDisplayHandle().
outputId	The display output ID. If "0", then the default outputId from NvAPI_GetAssociatedDisplayOutputId()) will be used.

Output Parameter

pInfo	The monitor and GPU's HDMI support info. See NV_HDMI_SUPPORT_INFO Struct.

Return Status[i]

NVAPI_OK	The request is completed.
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_INVALID_ARGUMENT	Invalid input parameter.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NV_HDMI_SUPPORT_INFO Struct

```
typedef struct
{
   NvU32      version;                        // structure version
    NvU32      isGpuHDMICapable       : 1;   // if the GPU can
handle HDMI
    NvU32      isMonUnderscanCapable  : 1;   // if the monitor
supports underscan
    NvU32      isMonBasicAudioCapable : 1;   // if the monitor
supports basic audio
```

NV_HDMI_SUPPORT_INFO Struct

```
    NvU32      isMonYCbCr444Capable   : 1;  // if YCbCr 4:4:4 is
supported
    NvU32      isMonYCbCr422Capable   : 1;  // if YCbCr 4:2:2 is
supported
    NvU32      isMonHDMI              : 1;  // if the monitor is
HDMI (with IEEE's HDMI registry ID)
    NvU32      EDID861ExtRev;               // the revision
number of the EDID 861 extension
 } NV_HDMI_SUPPORT_INFO;
```

NvAPI_GetInfoFrame()

OS/architecture : Windows XP / 32-bit and 64-bit
Earliest ForceWare Version: 97.00

This function returns the current infoframe data on the specified display.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GetInfoFrame(
                NvDisplayHandle   hNvDisplay,
                NvU32             outputId,
                NV_INFOFRAME_TYPE type,
                NV_INFOFRAME      *pInfoFrame);
```

Input Parameter

hNvDisplay	The NVIDIA display handle. It can be NVAPI_DEFAULT_HANDLE or a handle enumerated from NvAPI_EnumNvidiaDisplayHandle().
outputId	The display output id. If "0", then the default outputId from NvAPI_GetAssociatedDisplayOutputId()) is used.
type	The type of infoframe to set. See NV_INFOFRAME_TYPE Enum.

Output Parameter

pInfoFrame	The infoframe data, NULL means reset to the default value. See NV_INFOFRAME Struct.

Return Status[i]

NVAPI_OK	The request is completed.correctly set.
NVAPI_ERROR	Miscellaneous occurred.
NVAPI_INVALID_ARGUMENT	Invalid input parameter

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NV_INFOFRAME_TYPE Enum

```
typedef enum _NV_INFOFRAME_TYPE
{
    NV_INFOFRAME_TYPE_AVI   = 2,
    NV_INFOFRAME_TYPE_SPD   = 3,
    NV_INFOFRAME_TYPE_AUDIO = 4,
    NV_INFOFRAME_TYPE_MS    = 5,
} NV_INFOFRAME_TYPE;
```

NV_INFOFRAME_HEADER Struct

```
typedef struct
{
    NvU8 type;
    NvU8 version;
    NvU8 length;
} NV_INFOFRAME_HEADER;
```

At this time, the API is for the Windows OS, so NVAPI uses the following bit little endian definition to handle the translation

NV_AUDIO_INFOFRAME Struct

```
typedef struct
{
    // byte 1
    NvU8 channelCount     : 3;
    NvU8 rsvd_bits_byte1  : 1;
    NvU8 codingType       : 4;

    // byte 2
    NvU8 sampleSize       : 2;
    NvU8 sampleRate       : 3;
    NvU8 rsvd_bits_byte2  : 3;

    // byte 3
    NvU8  byte3;

    // byte 4
    NvU8  speakerPlacement;

    // byte 5
    NvU8 rsvd_bits_byte5  : 3;
    NvU8 levelShift       : 4;
    NvU8 downmixInhibit   : 1;

    // byte 6~10
    NvU8 rsvd_byte6;
    NvU8 rsvd_byte7;
    NvU8 rsvd_byte8;
    NvU8 rsvd_byte9;
    NvU8 rsvd_byte10;

}NV_AUDIO_INFOFRAME;
```

NV_VIDEO_INFOFRAME Struct

```
typedef struct
{
    // byte 1
    NvU8 scanInfo                  : 2;
    NvU8 barInfo                   : 2;
    NvU8 activeFormatInfoPresent : 1;
    NvU8 colorSpace                : 2;
    NvU8 rsvd_bits_byte1           : 1;

    // byte 2
    NvU8 activeFormatAspectRatio : 4;
    NvU8 picAspectRatio            : 2;
    NvU8 colorimetry               : 2;

    // byte 3
    NvU8 nonuniformScaling         : 2;
    NvU8 rsvd_bits_byte3           : 6;

    // byte 4
    NvU8 vic                       : 7;
    NvU8 rsvd_bits_byte4           : 1;

    // byte 5
    NvU8 pixelRepeat               : 4;
    NvU8 rsvd_bits_byte5           : 4;

    // byte 6~13
    NvU8 topBarLow;
    NvU8 topBarHigh;
    NvU8 bottomBarLow;
    NvU8 bottomBarHigh;
    NvU8 leftBarLow;
    NvU8 leftBarHigh;
    NvU8 rightBarLow;
    NvU8 rightBarHigh;

} NV_VIDEO_INFOFRAME;
```

NV_INFOFRAME Struct

```
typedef struct
```

NV_INFOFRAME Struct

```
{
    NV_INFOFRAME_HEADER    header;
                           // See NV_INFOFRAME_HEADER Struct
    union
    {
        NV_AUDIO_INFOFRAME audio;
                           // See  NV_AUDIO_INFOFRAME Struct
        NV_VIDEO_INFOFRAME video;
                           // See NV_VIDEO_INFOFRAME Struct
    }u;
} NV_INFOFRAME;
```

NvAPI_SetInfoFrame()

OS/architecture : Windows XP / 32-bit and 64-bit
Earliest ForceWare Version: 97.00

This function API returns the current infoframe data on the specified display.

Function Prototype

```
NVAPI_INTERFACE NvAPI_SetInfoFrame(
                NvDisplayHandle   hNvDisplay,
                NvU32             outputId,
                NV_INFOFRAME_TYPE type,
                NV_INFOFRAME      *pInfoFrame);
```

Input Parameter

hNvDisplay	NVIDIA Display selection. It can be NVAPI_DEFAULT_HANDLE or a handle enumerated from NvAPI_EnumNvidiaDisplayHandle().
outputId	The display output id. If "0", then the default outputId from NvAPI_GetAssociatedDisplayOutputId()) is used.
type	The type of infoframe to set. See NV_INFOFRAME_TYPE Enum.
pInfoFrame	The infoframe data, NULL means reset to the default value. See NV_INFOFRAME Struct.

Return Status[i]

NVAPI_OK	The request is completed.
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_INVALID_ARGUMENT	Invalid input parameter.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

DirectX Calls

This section describes the following API calls:

- NvAPI_D3D9_GetSurfaceHandle()
- NvAPI_D3D9_GetTextureHandle()
- NvAPI_D3D9_GpuSyncGetHandleSize()
- NvAPI_D3D9_GpuSyncInit()
- NvAPI_D3D9_GpuSyncEnd()
- NvAPI_D3D9_GpuSyncMapTexBuffer()
- NvAPI_D3D9_GpuSyncMapVertexBuffer()
- NvAPI_D3D9_GpuSyncAcquire()
- NvAPI_D3D9_GpuSyncRelease()
- NvAPI_D3D9_GetCurrentRenderTargetHandle()
- NvAPI_D3D9_GetCurrentZBufferHandle()
- NvAPI_D3D9_AliasPrimaryAsTexture()
- NvAPI_D3D9_PresentSurfaceToDesktop()
- NvAPI_D3D9_PresentVideo()
- NvAPI_D3D9_RestoreDesktop()
- NvAPI_D3D9_AliasPrimaryFromDevice()
- NvAPI_D3D9_SetResourceHint()
- NvAPI_D3D9_Lock()
- NvAPI_D3D9_Unlock()
- NvAPI_D3D9_LockForCUDA()

NvAPI_D3D9_GetSurfaceHandle()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function gets the handle of a given surface. This handle uniquely identifies the surface through all NvAPI entries.

Function Prototype

```
NvAPI_Status NvAPI_D3D9_GetSurfaceHandle(
             IDirect3DSurface9  *pSurface,
             NVDX_ObjectHandle  *pHandle);
```

Input Parameter

pSurface	Surface to be identified

Output Parameter

pHandle	The handle of the DirectX surface

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GetTextureHandle()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function gets the handle of a given DirectX 9 texture.

Function Prototype

```
NvAPI_Status NvAPI_D3D9_GetTextureHandle
               (IDirect3DTexture9  *pTexture,
                NVDX_ObjectHandle  *pHandle);
```

Input Parameter

pTexture	Surface to be identified

Output Parameter

pHandle	The handle of the DirectX 9 texture

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GpuSyncGetHandleSize()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 92.00 (R90), 95.40 (R95)

This function returns the size of the init and copy sync handles for the given Direct3D device. These handles are then allocated and initialized to zero by the application.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_GpuSyncGetHandleSize(
                IDirect3DDevice9 *pDev,
                unsigned int      *pInitHandleSize,
                unsigned int      *pMapHandleSize);
```

Input Parameter

pDev	Pointer to the graphics device

Output Parameter

pInitHandleSize	Pointer to the size of the GpuSync init handle
pMapHandleSize	Pointer to the size of the GpuSync copy handle

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GpuSyncInit()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 92.00 (R90), 95.40 (R95)

This function sets up sync functionality.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_GpuSyncInit(
                IDirect3DDevice9 *pDev,
                void *             syncInitData);
```

Input Parameter

pDev	Pointer to the graphics device
syncInitData	Sync initialization data

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GpuSyncEnd()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 92.00 (R90), 95.40 (R95)

This function tears down sync structures.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_GpuSyncEnd(
                IDirect3DDevice9 *pDev,
                void *             syncData);
```

Input Parameter

pDev	Pointer to the graphics device
syncData	Sync data

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GpuSyncMapTexBuffer()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 92.00 (R90), 95.40 (R95)

This function maps a texture to receive OpenGL data.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_GpuSyncMapTexBuffer(
                IDirect3DDevice9  *pDev,
                IDirect3DTexture9 *pTexture,
                void *             syncData);
```

Input Parameter

pDev	Pointer to the graphics device
pTexture	Pointer to the texture to map
syncData	Sync data

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GpuSyncMapVertexBuffer()

OS/architecture: Windows XP/32-bit and 64-bit,
Earliest ForceWare Version: 92.00 (R90), 95.40 (R95)

This function maps a vertex buffer to receive OpenGL data.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_GpuSyncMapVertexBuffer(
                IDirect3DDevice9       *pDev,
                IDirect3DVertexBuffer9 *pVertexBuffer,
                void *                  syncData);
```

Input Parameter

pDev	Pointer to the graphics device
pVertexBuffer	Pointer to the vertex buffer to map
syncData	Sync data

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GpuSyncAcquire()

OS/architecture: Windows XP/32-bit and 64-bit,
Earliest ForceWare Version: 92.00 (R90), 95.40 (R95)

This function acquires the semaphore for synchronization control of a mapped buffer.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_GpuSyncAcquire(
                IDirect3DDevice9 *pDev,
                void *            syncData);
```

Input Parameter

pDev	Pointer to the graphics device
syncData	accessMode - acquire mapped buffer read/write access

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GpuSyncRelease()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 92.00 (R90), 95.40 (R95)

This function releases the semaphore release for synchronization control of a mapped buffer.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_GpuSyncRelease(IDirect3DDevice9
*pDev,
                                              void * syncData);
```

Input Parameter

pDev	Pointer to the graphics device
syncData	accessMode - release mapped buffer read/write access

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GetCurrentRenderTargetHandle()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit,
Earliest ForceWare Version: 82.61

This function gets the handle of the current render target.

Function Prototype

```
NvAPI_Status NvAPI_D3D9_GetCurrentRenderTargetHandle(
             IDirect3DDevice9   *pDev,
             NVDX_ObjectHandle  *pHandle);
```

Input Parameter

pDev	Device for the current render target to be identified

Output Parameter

pHandle	The handle of the current render target

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_GetCurrentZBufferHandle()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function gets the handle of the current z-buffer.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_GetCurrentZBufferHandle
                 (IDirect3DDevice9  *pDev,
                  NVDX_ObjectHandle *pHandle);
```

Input Parameter

pDev	Device for the current z-buffer to be identified

Output Parameter

pHandle	The handle of the current z-buffer.

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_AliasPrimaryAsTexture()

OS/architecture : Windows XP / 32-bit and 64-bit,,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function creates a texture that is an alias of the current device's primary surface.

Function Prototype

```
NvAPI_Status NvAPI_D3D9_AliasPrimaryAsTexture(
             IDirect3DDevice9   *pDev,
             NvU32               dwIndex,
             IDirect3DTexture9 **ppTexture,
             NVDX_ObjectHandle  *pHandle = 0);
```

Input Parameter

pDev	The device to get primary surface from
dwIndex	The index to the primary flipchain of device (usually 0)

Output Parameter

ppTexture	Fill with the texture created
pHandle	If non-NULL, fill with the NVDX handle of the created texture

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_PresentSurfaceToDesktop()

OS/architecture : Windows XP / 32-bit and 64-bit, ,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function presents a given surface to the desktop. This interface can be used to start a full-screen flipping mode even within windowed Direct3D applications.

The application is responsible for determining which devices are available on the current clone configuration through NVCPL interfaces

Function Prototype

```
NvAPI_Status NvAPI_D3D9_PresentSurfaceToDesktop(
             IDirect3DDevice9   *pDev,
             NVDX_ObjectHandle   surfaceHandle,
             NvU32               dwFlipFlags,
             NvU32               dwExcludeDevices = 0);
```

Input Parameters

pDev	The target display device for the desktop.
surfaceHandle	The surface handle obtained from NVD3D9_GetSurfaceHandle NOTE: NVDX_OBJECT_NONE means restore.
dwFlipFlags	Flags to indicate SYNC mode (other bits reserved and must be 0)
dwExcludeDevices	This is a bitmask (usually 0) to indicate which device will be EXCLUDED from this presentation. This is only effective when used in a Clone mode configuration where the application wants to show the specially rendereed screen on one monitor and the normal desktop on the other monitor.

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

.

dwFlipFlags Options

```
#define NV_FLIPFLAG_VSYNC                      0x00000001
           // SYNCMODE  (bit 0:1)-             0:NOSYNC,
                                               1:VSYNC,
                                               2:HSYNC
#define NV_FLIPFLAG_HSYNC                      0x00000002
#define NV_FLIPFLAG_TRIPLEBUFFERING            0x00000004
           // TRIPLEBUFFERING  (bit 2)-  0: DoubleBuffer,
                                               1:TripleBuffer or more
```

NvAPI_D3D9_PresentVideo()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 91.10 (Rel90) / 95.06 (Rel95)

This function signals a frame as final, complete, and ready for presentation. The frame can optionally be rendered to the overlay, but the function should be called regardless of whether any actual rendering occurs. If the user has enabled full screen video in a multi-head mode, this frame will also be rendered on the secondary device.

Release 95 Implementation

Function Prototype (Rel95)

```
NVAPI_INTERFACE NvAPI_D3D9_PresentVideo(
                IDirect3DDevice9 *pDev,
                NVDX_ObjectHandle surfaceHandle,
                NvU32             dwPVFlags,
                NvU32             dwColourKey,
                NvU32             dwTimeStampLow,
                NvU32             dwTimeStampHigh,
                NvU32             dwFlipRate,
                NvU32             dwUnclippedSrcX,
                NvU32             dwUnclippedSrcY,
                NvU32             dwUnclippedSrcWidth,
                NvU32             dwUnclippedSrcHeight,
                NvU32             dwClippedSrcX,
                NvU32             dwClippedSrcY,
                NvU32             dwClippedSrcWidth,
                NvU32             dwClippedSrcHeight,
                NvU32             dwDstX,
                NvU32             dwDstY,
                NvU32             dwDstWidth,
                NvU32             dwDstHeight);
```

Input Parameters (Rel95)

pDev	The device (display) to present to
surfaceHandle	The surface handle obtained from NvAPI_D3D9_GetSurfaceHandle() or NvAPI_D3D9_GetCurrentRenderTargetHandle()
dwPVFlags	Presentation flags See "PresentVideo Flags (Rel95)" on page 79 for description)
dwColourKey	Colour key to use if NV_PVFLAG_DST_KEY is set
dwTimeStamp*	If NV_PVFLAG_USE_STAMP is set, time in ns when the frame is to be presented. If NV_PVFLAG_SET_STAMP is set, set the current time to this, and present on the next VBlank
dwFlipRate	Set to the current flip rate Set to zero if the frame to be presented is a still frame
dwUnclippedSrc*	Unclipped source rectangle of the entire frame of data

Input Parameters (Rel95) (continued)

dwClippedSrc*	Cropped source rectangle. It is the caller's responsibility to crop the source if the desktop crops the destination.
dwDst*	Destination rectangle (in desktop coordinates) of the overlay. It is the caller's responsibility to crop the destination against the desktop.

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

PresentVideo Flags (Rel95)

```
#define NV_PVFLAG_ODD           0x00000001
 // Field is odd
#define NV_PVFLAG_EVEN          0x00000002
 // Field is even
#define NV_PVFLAG_PROTECTED     0x00000004
 // Indicates that this frame is protected and guarantees the
    full-screen video will not display this frame on any
    secondary device.
    Conversely, not setting this indicates an unprotected frame.
#define NV_PVFLAG_PROGRESSIVE   0x00000008
 // Indicates a progressive frame. If the odd or even flags are
    set in conjunction with this, it indicates the original
    field that generated this deinterlaced frame, and attempts
    to synchronize this presentation to the corresponding
    display field of an interlaced display
#define NV_PVFLAG_SHOW          0x00000010
 // Show the overlay. If the application is minimized or
    obscured, continue to call NvAPI_D3D9_PresentVideo for every
    complete frame without this flag set.
    If enabled, the unprotected video will continue to play full
    screen on the secondary device, using the pixel aspect
    cached from the last time a frame was shown. To change the
    pixel aspect while hidden, the caller must "show" a frame at
    least once with a new clipped source and destination
    rectangle. This shown frame can be rendered invisible with
    appropriate selection of colour key.
#define NV_PVFLAG_FAST_MOVE     0x00000020
 // Move overlay position without waiting for VBlank. The only
    parameters used are dwDstX, dwDstY, and NV_PVFLAG_SHOW.
#define NV_PVFLAG_WAIT          0x00000040
 // If set, indicates a blocking flip - wait until flip queue
    can accept another flip. A non-blocking flip will return an
    error if the flip cannot be queued yet.
```

PresentVideo Flags (Rel95)

```
#define NV_PVFLAG_DST_KEY       0x00000100
 // Use destination colour key.
#define NV_PVFLAG_FULLSCREEN    0x00000200
  // Indicates that the overlay is playing fullscreen on the
     desktop.  This bit is used to automatically overscan the
     image on TV's.
#define NV_PVFLAG_SET_STAMP     0x00001000
  // Set the current time.
#define NV_PVFLAG_USE_STAMP     0x00002000
  // If set, use timestamps.
     If not set, flip on the next VBlank.
```

Release 90 Implementation

Function Prototype (Rel90)

```
NVAPI_INTERFACE NvAPI_D3D9_PresentVideo(
                IDirect3DDevice9           *pDev,
                NV_DX_PRESENT_VIDEO_PARAMS *pPVParams);
```

Input Parameters (Rel90)

pDev	The device (display) to present to
pPVParams	Present video parameters See The surface handle obtained from NvAPI_D3D9_GetSurfaceHandle() or NvAPI_D3D9_GetCurrentRenderTargetHandle()

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NV_DX_PRESENT_VIDEO_PARAMS1 Struct (Rel90)

```
typedef struct
{
    NvU32 version;
    NVDX_ObjectHandle surfaceHandle;
    NvU32 pvFlags;
    NvU32 colourKey;
    NvU32 timeStampLow;
    NvU32 timeStampHigh;
    NvU32 flipRate;
    NvSBox srcUnclipped;
    NvSBox srcClipped;
    NvSBox dst;
} NV_DX_PRESENT_VIDEO_PARAMS1;
```

NvAPI_D3D9_RestoreDesktop()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This is not an interface, but rather a short-hand helper.

Function Prototype

```
inline int NvAPI_D3D9_RestoreDesktop
           (IDirect3DDevice9 *pDev)
{
```

Function Prototype

```
return NvAPI_D3D9_PresentSurfaceToDesktop
           (pDev,
            NVDX_OBJECT_NONE,
            0);
}
```

Return Status

One of the NvAPI status codes (see“NvAPI Return Status Codes” on page 13).

NvAPI_D3D9_AliasPrimaryFromDevice()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function creates an alias surface from the given pDevFrom's primary swap chain.

Function Prototype

```
NvAPI_Status NvAPI_D3D9_AliasPrimaryFromDevice(
             IDirect3DDevice9   *pDevTo,
             IDirect3DDevice9   *pDevFrom,
             NvU32               dwIndex,
             IDirect3DSurface9 **ppSurf,
             NVDX_ObjectHandle  *pHandle = 0);
```

Input Parameters

pDevTo	Where the new surfaces are created
pDevFrom	Where the surfaces are aliased from
dwIndex	Index to the primary flipchain of pDevFrom

Output Parameters

ppSurf	Filled with new surface pointer (to be released by the caller)
pHandle	(optional) If non-NULL, filled with SurfaceHandle of the surface. The same can be achieved by calling NVD3D9_GetSurfaceHandle afterwards.

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

NvAPI_D3D9_SetResourceHint()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 96.40

This is a general purpose function for passing down various resource related hints to the driver. Hints are divided into categories and types within each category.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_SetResourceHint(
                IDirect3DDevice9               *pDev,
                NVDX_ObjectHandle               obj,
                NVAPI_SETRESOURCEHINT_CATEGORY dwHintCategory,
                NvU32                           dwHintName,
                NvU32                          *pdwHintValue);
```

Input Parameters

pDevTo	A valid device context
obj	The previously obtained HV resource handle
dwHintCategory	The hint category See NVAPI_SETRESOURCEHINT_CATEGORY.
dwHintName	The hint within the category **dwHintCategory** See "Available Hints" on page 85,
pdwHintValue	Pointer to the location containing the hint value

Output Parameters

dwHintValue	The value of the previous hint

Return Status

One of the NvAPI status codes (see"NvAPI Return Status Codes" on page 13).

Available Categories

NVAPI_SETRESOURCEHINT_CATEGORY

```
typedef enum _NVAPI_SETRESOURCEHINT_CATEGORY
{
    NvApiHints_Sli = 1,
}  NVAPI_SETRESOURCEHINT_CATEGORY;
```

Available Hints

NVAPI_SETRESOURCEHINT_SLI_HINTS

```
typedef enum _NVAPI_SETRESOURCEHINT_SLI_HINTS
{
    NvApiHints_Sli_InterfarameAwareForTexturing = 1,
}  NVAPI_SETRESOURCEHINT_SLI_HINTS;
```

InterframeAwareForTexturing tells the driver to discard any interframe dirty state (skip inter-GPU transfers) on this object when texturing from it. Other operations (clear, render, blit, CPU lock) are not affected by this bit.

The value of the bit is the maximum difference (in frames) of the last modifications of the current and golden-copy GPUs which is allowed to get discarded. For example, if the value is 1, then the inactive buffer will be discarded only if it was modified on the same or previous frame and will be transferred to the active GPU otherwise. The default value is zero.

NvAPI_D3D9_Lock()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function locks a given surface identified by the specified handle.

This function can provide CPU access to all objects including render targets, z-buffers, textures, vertex buffers, and index buffers.

- If an object can be accessed using standard DirectX 9 calls, do not use this function.
- Lock should be called right before accessing the CPU.
- Any 3D rendering or state change may cause the locked surface to be lost. When that happens, trying to access the cached CPU address may cause the application to crash.

Function Prototype

```
NvAPI_Status NvAPI_D3D9_Lock
                (IDirect3DDevice9   *pDev,
                 NVDX_ObjectHandle   obj,
                 NvU32               dwLockFlags,
                 void              **ppAddress,
                 NvU32              *pPitch);
```

Input Parameters

pDev	The Direct3D device
obj	The NVIDIA DirectX object handle
dwLockFlags	One of the flags listed in “dwLockFlags” on page 86.

Output Parameters

ppAddress	Pointer to the ???
pPitch	Pointer to ???

Return Status

One of the NvAPI status codes (see“NvAPI Return Status Codes” on page 13).

dwLockFlags

```
#define NV_ACCESSFLAG_READONLY  0x00000001
```

dwLockFlags

```
#define NV_ACCESSFLAG_DISCARD   0x00000002
```

NvAPI_D3D9_Unlock()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 82.61

This function unlocks a given surface identified by the specified object handle.

This function can provide CPU access to all objects including render targets, z-buffers, textures, vertex buffers, and index buffers.

- If an object can be accessed using standard DirectX 9 calls, do not use this function.
- Unlock should be called right after accessing the CPU.
- Any 3D rendering or state change may cause the locked surface to be lost. When that happens, trying to access the cached CPU address may cause the application to crash.

Function Prototype

```
NvAPI_Status NvAPI_D3D9_Unlock
               (IDirect3DDevice9   *pDev,
                NVDX_ObjectHandle   obj);
```

Input Parameters

pDev	The Direct3D device
obj	The resource handle

Return Status

One of the NvAPI status codes (see“NvAPI Return Status Codes” on page 13).

NvAPI_D3D9_LockForCUDA()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 96.40

This function locks a given surface identified by the handle, and can provide access to all objects, including render targets, z-buffers, textures, vertex buffers, and index buffers. NVAPI_D3D9_ LockForCUDA() is analogous to a LockForCPU type call, except that the memory is read/written by CUDA (compute running in a separate channel) as opposed to the CPU. It should be called right before giving access to CUDA and Unlock called right after the access is achieved.

Function Prototype

```
NVAPI_INTERFACE NvAPI_D3D9_LockForCUDA(
                IDirect3DDevice9 *pDev,
                NVDX_ObjectHandle obj,
                NvU32             dwLockFlags,
                NvU32            *pClient,
                NvU32            *pHandle,
                NvU64            *pOffset,
                NvU32            *pSize);
```

Input Parameters

pDev	A valid device context
obj	The resource handle
dwLockFlags	
pClient	
pHandle	
pOffset	
pSize	

Return Status

One of the NvAPI status codes (see“NvAPI Return Status Codes” on page 13).

GPU Clock Control Calls

The APIs in this section allow the user to get and set individual clock domains on a per-GPU basis:

- "NvAPI_GPU_GetPerfClocks()" on page 91
- "NvAPI_GPU_SetPerfClocks()" on page 92

Performance Table Overclocking Defines and Structure

NV_GPU_PERF_CLOCK_DOMAIN_ID

```
typedef enum _NV_GPU_PERF_CLOCK_DOMAIN_ID
{
    NVAPI_GPU_PERF_CLOCK_DOMAIN_NV      = 0,
    NVAPI_GPU_PERF_CLOCK_DOMAIN_M       = 4,

} NV_GPU_PERF_CLOCK_DOMAIN_ID;
```

```
#define NVAPI_MAX_GPU_PERF_CLOCKS       32
```

```
#define NVAPI_MAX_PERF_CLOCK_LEVELS                 12
#define NVAPI_TARGET_ALL_PERF_LEVELS                0xffffffff
#define NV_PERF_CLOCK_LEVEL_STATE_DEFAULT           0x00000000
        // Level is in its default state
#define NV_PERF_CLOCK_LEVEL_STATE_OVERCLOCKED       0x00000001
        // Level is overclocked
#define NV_PERF_CLOCK_LEVEL_STATE_DESKTOP           0x00000002
        // 2D desktop perf level
#define NV_PERF_CLOCK_LEVEL_STATE_PERFORMANCE       0x00000004
        // 3D applications perf level
#define NV_PERF_CLOCK_LEVEL_STATE_TEST              0x00000008
        // Test the new clocks for this level.
            Does not apply.
#define NV_PERF_CLOCK_LEVEL_STATE_TEST_SUCCESS      0x00000010
        // Test result
#define NV_PERF_CLOCK_GPU_STATE_DEFAULT             0x00000000
        // Default state
#define NV_PERF_CLOCK_GPU_STATE_DYNAMIC_SUPPORTED   0x00000001
        // GPU supports dynamic performance level transitions
#define NV_PERF_CLOCK_GPU_STATE_DESKTOP             0x00000002
        // GPU in desktop level
```

```
#define NV_PERF_CLOCK_GPU_STATE_PERFORMANCE          0x00000004
        // GPU in performance level
#define NV_PERF_CLOCK_GPU_STATE_ACTIVE_CLOCKING_SUPPORTED
         0x00000008     // Active clocking supported
#define NV_PERF_CLOCK_GPU_STATE_ACTIVE_CLOCKING_ENABLE
         0x00000010     // Enable active clocking
#define NV_PERF_CLOCK_GPU_STATE_ACTIVE_CLOCKING_DISABLE
         0x00000020     // Disable active clocking
#define NV_PERF_CLOCK_GPU_STATE_MEMCLK_CONTROL_DISABLED
         0x00000040     // Mmemory clock control disabled
#define NV_PERF_CLOCK_GPU_STATE_GFXCLK_CONTROL_DISABLED
         0x00000080     // Ccore clock control disabled
#define NV_PERF_CLOCK_GPU_STATE_SET_DEFERRED
         0x00000100     // No immediate perf transitions.
                           Deferred until perf triggers kick in.
```

NV_GPU_PERF_CLOCK_TABLE Struct

```
typedef struct
{
    NvU32   version;     // [IN]Perf clock table version
    NvU32   levelCount;  // Number of the performance levels
                         The count increases every time a level
                         is overclocked.
    NvU32   gpuPerflevel; //[OUT] The current perf level.
                          This is a dynamic level which can
                          possibly change on every call.
    NvU32   domainCount;  //[IN/OUT] The number of domains
    NvU32   gpuPerfFlags; //[IN/OUT] GPU flags - one of the
                         flags defined in NV_PERF_CLOCK_GPU_STATE.
    struct
    {
       NvU32   level;  //[IN/OUT] Performance level indicator,
                           Range 0 to levelCount - 1.
        NvU32   flags;  //[IN/OUT] Perf level flags - one or more
                      flags defined in NV_PERF_CLOCK_LEVEL_STATE.
       struct
        {
          NV_GPU_PERF_CLOCK_DOMAIN_ID  domainId;     ///[IN/OUT]
                      The current domain indicator - one of
                      the IDs from NV_GPU_CLOCK_DOMAIN_ID
          NvU32      domainFlags; /// Reserved unused domain
                                        flags
          NvU32       currentFreq;   ///[IN/OUT] current clock KHz
          NvU32      defaultFreq;  /// Default clock (KHz)
          NvU32      minFreq;      /// Min KHz
```

NV_GPU_PERF_CLOCK_TABLE Struct

```
          NvU32      maxFreq;      /// Max KHz
          NvU32      bSetClock:1;  ///[IN] If set during
                             NvAPI_GPU_SetPerfClocks call,
                             this domain currentFreq is applied.
        } domain[NVAPI_MAX_GPU_PERF_CLOCKS];
    } perfLevel[NVAPI_MAX_PERF_CLOCK_LEVELS];

} NV_GPU_PERF_CLOCK_TABLE;
```

NvAPI_GPU_GetPerfClocks()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 88.90

NV_GPU_PERF_CLOCK_TABLE Structure: Version 1

This function retrieves the performance clock table information for one or all of the supported levels.

Use **NV_GPU_PER_CLOCK_TABLE_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetPerfClocks(
                NvPhysicalGpuHandle      hPhysicalGpu,
                NvU32                    level,
                NV_GPU_PERF_CLOCK_TABLE *pPerfClkTable);
```

Input Parameter

hPhysicalGPU	The handle for the physical GPU
level	Specific level selection. Zero for all levels. The number of levels increases with overclocking of the levels.

Output Parameter

pPerfClkTable	[OUT] Table of performance levels retrieved. See NV_GPU_PERF_CLOCK_TABLE Struct.

Return Status[i]

NVAPI_OK	Request has been completed.
NVAPI_ERROR	A Miscellaneous error occured.
NVAPI_HANDLE_INVALIDATED	The handle passed has been invalidated.

Return Status[i]

NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	The handle passed is not a physical GPU handle
NVAPI_INCOMPATIBLE_STRUCT_VERSION	The version of the INFO struct is not supported.

i. See “NvAPI Return Status Codes” on page 13 for a list of other possible return codes.

NvAPI_GPU_SetPerfClocks()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 88.90

NV_GPU_PERF_CLOCK_TABLE Structure: Version 1

This function overclocks a specific level in the performance table or overclock all levels with **bSetClock** set.

Use **NV_GPU_PER_CLOCK_TABLE_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_SetPerfClocks(
                NvPhysicalGpuHandle      hPhysicalGpu,
                NvU32                    level,
                NV_GPU_PERF_CLOCK_TABLE *pPerfClkTable);
```

Input Parameter

hPhysicalGPU	The handle for the physical GPU
level	Specific level selection. Zero for all levels. The number of levels increases with overclocking of the levels.
pPerfClkTable	Table of performance levels to set. Any other than DEFAULT for GPU and Level flags - gpuPerfFlags and level flags gets applied. If bSetClock is set, currentFreq gets applied. Overclocking DOMAIN_NV requires simultaneous overclocking of DOMAIN_M, otherwise overclocking will fail. See NV_GPU_PERF_CLOCK_TABLE Struct.

Return Status[i]

NVAPI_OK	Request has been completed.
NVAPI_ERROR	A Miscellaneous error occured.
NVAPI_HANDLE_INVALIDATED	The handle passed has been invalidated.

Return Status[i]

`NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE`	The handle passed is not a physical GPU handle
`NVAPI_INCOMPATIBLE_STRUCT_VERSION`	The version of the INFO struct is not supported.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

OpenGL Related Calls

This section describes the following OpenGL configuration calls:

- "NvAPI_OGL_ExpertModeGet() / NvAPI_OGL_ExpertModeSet()" on page 94
- "NvAPI_OGL_ExpertModeDefaultsSet() / NvAPI_OGL_ExpertModeDefaultsGet()" on page 96

NvAPI_OGL_ExpertModeGet() / NvAPI_OGL_ExpertModeSet()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 84.11, 88.00

This function configures OpenGL Expert Mode, an API usage feedback and advice reporting mechanism. This call affects the settings only for the current process. The settings are reset to the defaults when the process quits.

These functions are valid only for the current OpenGL context. Calling these functions prior to creating a context and then making it current (calling **`MakeCurrent()`**) will result in errors and undefined behavior.

Function Prototype

```
NVAPI_INTERFACE NvAPI_OGL_ExpertModeGet(
                NvU32                    *pexpertDetailLevel,
                NvU32                    *pexpertReportMask,
                NvU32                    *pexpertOutputMask);
                NVAPI_OGLEXPERT_CALLBACK *pexpertCallback);
```

Function Prototype

```
NVAPI_INTERFACE NvAPI_OGL_ExpertModeSet(
                NvU32                    expertDetailLevel,
                NvU32                    expertReportMask,
                NvU32                    expertOutputMask);
                NVAPI_OGLEXPERT_CALLBACK expertCallback);
```

Input Parameter

expertDetailLevel	Value which specifies the detail level in the feedback stream. Set the detail level to anything greater than zero to enable the output. A value of zero disables the output. Meaningful values range from 0-30 in increments of 10.
expertReportMask	Mask, made up of NVAPI_OGLEXPERT_REPORT bits, that specifies the areas of functional interest.
expertOutputMask	Mask, made up of NVAPI_OGLEXPERT_OUTPUT bits, that specifies the feedback output location.
expertCallback	This is a simple callback function to obtain the feedback stream. The **OUTPUT_TO_CALLBACK** bit must be set in NVAPI_OGLEXPERT_OUTPUT. The function is called once per each fully-qualified feedback stream entry.

Output Parameter

pexpertDetailLevel	Pointer to the detail level in the feedback stream. A value of zero indicates that the output is disabled.
pexpertReportMask	Pointer to the NVAPI_OGLEXPERT_REPORT bits that specify functional areas.
pexpertOutputMask	Pointer to the NVAPI_OGLEXPERT_OUTPUT bits that specify the feedback output location.
pexpertCallback	Pointer to the callback function to obtain the feedback stream. The **OUTPUT_TO_CALLBACK** bit must be set in NVAPI_OGLEXPERT_OUTPUT. The function is called once per each fully-qualified feedback stream entry.

Return Status[i]

NVAPI_OK	
NVAPI_ERROR	OpenGL driver failed to load properly.
NVAPI_API_NOT_INITIALIZED	
NVAPI_NVIDIA_DEVICE_NOT_FOUND	
NVAPI_OPENGL_CONTEXT_NOT_CURRENT	No NVIDIA OpenGL context which supports GLExpert has been made current.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NVAPI_OGLEXPERT_REPORT

```
#define NVAPI_OGLEXPERT_REPORT_NONE          0x00000000
#define NVAPI_OGLEXPERT_REPORT_ERROR         0x00000001
#define NVAPI_OGLEXPERT_REPORT_SWFALLBACK    0x00000002
#define NVAPI_OGLEXPERT_REPORT_PROGRAM       0x00000004
#define NVAPI_OGLEXPERT_REPORT_VBO           0x00000008
#define NVAPI_OGLEXPERT_REPORT_FBO           0x00000010
#define NVAPI_OGLEXPERT_REPORT_ALL           0xFFFFFFFF
```

NVAPI_OGLEXPERT_OUTPUT

```
#define NVAPI_OGLEXPERT_OUTPUT_TO_CONSOLE    0x00000001
#define NVAPI_OGLEXPERT_OUTPUT_TO_DEBUGGER   0x00000004
#define NVAPI_OGLEXPERT_OUTPUT_TO_CALLBACK   0x00000008
#define NVAPI_OGLEXPERT_OUTPUT_TO_ALL        0xFFFFFFFF
```

NvAPI_OGL_ExpertModeDefaultsSet() / NvAPI_OGL_ExpertModeDefaultsGet()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 84.11, 88.00

This function configures OpenGL Expert Mode global default settings. These settings apply to any OpenGL application that opens after this call is made. They do not apply to applications that are running at the time the call is made.

Function Prototype

```
NVAPI_INTERFACE NvAPI_OGL_ExpertModeDefaultsSet(
                NvU32 expertDetailLevel,
                NvU32 expertReportMask,
                NvU32 expertOutputMask);
```

Function Prototype

```
NVAPI_INTERFACE NvAPI_OGL_ExpertModeDefaultsGet(
                NvU32 *pexpertDetailLevel,
                NvU32 *pexpertReportMask,
                NvU32 *pexpertOutputMask);
```

Input Parameter

expertDetailLevel	Value which specifies the detail level in the feedback stream. Set the detail level to anything greater than zero to enable the output. A value of zero disables the output. Meaningful values range from 0-30 in increments of 10.
expertReportMask	Mask made up of NVAPI_OGLEXPERT_REPORT bits, this parameter specifies the areas of functional interest.
expertOutputMask	Mask made up of NVAPI_OGLEXPERT_OUTPUT bits, this parameter specifies the feedback output location.

Output Parameter

pexpertDetailLevel	Pointer to the detail level in the feedback stream. A value of zero indicates that the output is disabled.
pexpertReportMask	Pointer to the NVAPI_OGLEXPERT_REPORT bits that specify the functional areas.
pexpertOutputMask	Pointer to the NVAPI_OGLEXPERT_OUTPUT bits that specify the feedback output location.

Return Status[i]

NVAPI_OK
NVAPI_ERROR
NVAPI_API_NOT_INITIALIZED

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NVAPI_OGLEXPERT_REPORT

```
#define NVAPI_OGLEXPERT_REPORT_NONE          0x00000000
#define NVAPI_OGLEXPERT_REPORT_ERROR         0x00000001
#define NVAPI_OGLEXPERT_REPORT_SWFALLBACK    0x00000002
#define NVAPI_OGLEXPERT_REPORT_PROGRAM       0x00000004
#define NVAPI_OGLEXPERT_REPORT_VBO           0x00000008
#define NVAPI_OGLEXPERT_REPORT_FBO           0x00000010
#define NVAPI_OGLEXPERT_REPORT_ALL           0xFFFFFFFF
```

NVAPI_OGLEXPERT_OUTPUT

```
#define NVAPI_OGLEXPERT_OUTPUT_TO_CONSOLE    0x00000001
#define NVAPI_OGLEXPERT_OUTPUT_TO_DEBUGGER   0x00000004
#define NVAPI_OGLEXPERT_OUTPUT_TO_CALLBACK   0x00000008
#define NVAPI_OGLEXPERT_OUTPUT_TO_ALL        0xFFFFFFFF
```

I^{2}C Calls

The calls in this section provided the ability to read or write data using the I2C protocol.

- "NvAPI_I2CRead()" on page 99
- "NvAPI_I2CWrite()" on page 100

I2C API Data Structures and Enums

This following are structures that are used by other API calls in this section.

NV_I2C_INFO Structure

```
typedef struct
{
    NvU32  version;         //Structure version
    NvU32  displayMask;     //The display mask of the target
                              display.
    NvU8   bIsDDCPort;      //Flag indicating whether the I2C
                              traffic is intended for DDC/CI or
                              non-DDC/CI. This lets the API
                              know how to use the port
                              (DDC or communications).
                               Set to 1, as only DDC is supported
                               at this time.
    NvU8   i2cDevAddress;   //The I2C target device address
    NvU8*  pbI2cRegAddress;//The I2C target register address
    NvU32  regAddrSize;     //The size, in bytes, of the
                              target register address
    NvU8*  pbData;          //The buffer of data which is to
                              be read or written
    NvU32  cbSize;          //The size of the data buffer to be
                              read or written.
    NvU32  i2cSpeed;        //The requested speed at which the
                              transaction is to occur,
                              between 28kbs and 40kbps:
          Higher speeds can be requested, but the driver will use
          the maximum reliable speed between 28kbps and 40kbps.
          A future driver version will support higher speeds.
 } NV_I2C_INFO;
```

NvAPI_I2CRead()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.90

NV_I2C_INFO Structure: Version 1

This function reads the data buffer from the I2C port.

Use **NV_I2C_INFO_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_I2CRead(
                NvPhysicalGpuHandle  hPhysicalGpu,
                NV_I2C_INFO         *pI2cInfo);
```

Input Parameter

hPhysicalGPU	NVIDIA physical GPU handle.
pI2cInfo	Pointer to the NV_I2C_INFO Structure.

Return Status[i]

NVAPI_OK	Request completed.
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_HANDLE_INVALIDATED	Handle passed has been invalidated.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	Handle passed is not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_I2CWrite()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.90

NV_I2C_INFO Structure: Version 1

This function writes the data buffer to the I2C port.

Use **NV_I2C_INFO_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_I2CWrite(
                NvPhysicalGpuHandle  hPhysicalGpu,
                NV_I2C_INFO         *pI2cInfo);
```

Input Parameter

hPhysicalGPU	NVIDIA physical GPU handle.
pI2cInfo	Pointer to the NV_I2C_INFO Structure.

Return Status[i]

NVAPI_OK	Request completed.
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_HANDLE_INVALIDATED	Handle passed has been invalidated.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	Handle passed is not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

GPU Cooler Calls

The APIs in this section get and set the fan level or equivalent cooler levels for various target devices associated with the GPU.

Cooler Defines, Enumerations, and Structures

```
#define NVAPI_MAX_COOLERS_PER_GPU   3
#define NVAPI_MIN_COOLER_LEVEL      0
#define NVAPI_MAX_COOLER_LEVEL      100
#define NVAPI_MAX_LEVELS_PER_POLICY 24
```

NV_COOLER_TYPE Enum

```
typedef enum
 {
    NVAPI_COOLER_TYPE_NONE = 0,
    NVAPI_COOLER_TYPE_FAN,
    NVAPI_COOLER_TYPE_WATER,
    NVAPI_COOLER_TYPE_LIQUID_NO2,
 } NV_COOLER_TYPE;
```

NV_COOLER_CONTROLLER Enum

```
typedef enum
{
    NVAPI_COOLER_CONTROLLER_NONE = 0,
    NVAPI_COOLER_CONTROLLER_ADI,
    NVAPI_COOLER_CONTROLLER_INTERNAL,
} NV_COOLER_CONTROLLER;
```

NV_COOLER_POLICY Enum

```
typedef enum
{
    NVAPI_COOLER_POLICY_NONE = 0,
    NVAPI_COOLER_POLICY_MANUAL,  //Manual adjustment of cooler
                                   level. Gets applied right away
                                  independent of temperature or
                                  performance level.
    NVAPI_COOLER_POLICY_PERF,    //GPU performance controls
                                   the cooler level.
    NVAPI_COOLER_POLICY_TEMPERATURE_DISCRETE = 4,
                                 //Discrete thermal levels
                                   control the cooler level.
    NVAPI_COOLER_POLICY_TEMPERATURE_CONTINUOUS = 8,
                                  //Cooler level adjusted at
                                    continuous thermal levels.
    NVAPI_COOLER_POLICY_HYBRID,   //Hybrid of performance and
                                    temperature levels.
} NV_COOLER_POLICY;
```

NV_COOLER_TARGET Enum

```
typedef enum
{
    NVAPI_COOLER_TARGET_NONE = 0,
    NVAPI_COOLER_TARGET_GPU,
    NVAPI_COOLER_TARGET_MEMORY,
    NVAPI_COOLER_TARGET_POWER_SUPPLY = 4,
    NVAPI_COOLER_TARGET_ALL = 7    //This cooler cools all of
                                     the components related
                                     to its target GPU.
 } NV_COOLER_TARGET;
```

NV_COOLER_CONTROL Enum

```
typedef enum
{
    NVAPI_COOLER_CONTROL_NONE = 0,
    NVAPI_COOLER_CONTROL_TOGGLE,   //ON/OFF
    NVAPI_COOLER_CONTROL_VARIABLE, //Supports variable control.
} NV_COOLER_CONTROL;
```

NV_COOLER_ACTIVITY_LEVEL Enum

```
typedef enum
{
   NVAPI_INACTIVE = 0,   //Inactive or unsupported
   NVAPI_ACTIVE    = 1,  //Active and spinning in case of fan
} NV_COOLER_ACTIVITY_LEVEL;
```

->

NV_GPU_GETCOOLER_SETTINGS Structure

```
typedef struct
{
    NvU32   version;      //Structure version
    NvU32   count;        //Number of coolers associated with
                            the selected GPU
    struct
    {
        NV_COOLER_TYPE    type;     //Type of cooler
                                 See "NV_COOLER_TYPE Enum" on
page 101.
        NV_COOLER_CONTROLLER controller;    //internal, ADI...
                      See "NV_COOLER_CONTROLLER Enum" on page 101.
        NvU32             defaultMinLevel; //the min default
                                         value % of the cooler
        NvU32             defaultMaxLevel; //the max default
                                         value % of the cooler
         NvU32              currentMinLevel; //the current allowed
                                                 min value % of the
                                                 cooler
         NvU32              currentMaxLevel; //the current allowed
                                                  max value % of the
                                                 cooler
        NvU32             currentLevel;   //the current value %
                                               of the cooler
        NV_COOLER_POLICY  defaultPolicy;  //Default cooler
                                             control policy
                           See "NV_COOLER_POLICY Enum" on page 102.
        NV_COOLER_POLICY   currentPolicy;  //Current cooler
                                             control policy
                           See "NV_COOLER_POLICY Enum" on page 102.
        NV_COOLER_TARGET   target;       //Cooler target -
                            See "NV_COOLER_TARGET Enum" on page 102.
        NV_COOLER_CONTROL  controlType;  //Toggle or variable -
                                 See "NV_COOLER_CONTROL Enum" on
page 102.
```

NV_GPU_GETCOOLER_SETTINGS Structure

```
        NV_COOLER_ACTIVITY_LEVEL  active; //
                    See "NV_COOLER_ACTIVITY_LEVEL Enum" on
page 103.
    } cooler[NVAPI_MAX_COOLERS_PER_GPU];

 } NV_GPU_GETCOOLER_SETTINGS;
```

NV_GPU_SETCOOLER_LEVEL Structure

```
 typedef struct
 {
     NvU32   version;        //structure version
     struct
     {
         NvU32            currentLevel;  //The new value % of
                                           the cooler
        NV_COOLER_POLICY  currentPolicy; //The new cooler
                                           control policy
                             See "NV_COOLER_POLICY Enum" on
page 102.
     } cooler[NVAPI_MAX_COOLERS_PER_GPU];

 } NV_GPU_SETCOOLER_LEVEL;
```

NV_GPU_COOLER_POLICY_TABLE Structure

```
 typedef struct
 {
     NvU32               version;    //structure version
     NV_COOLER_POLICY    policy;     //Selected policy to
                                     update the cooler levels
                  See "NV_COOLER_POLICY Enum" on page 102.
     struct
     {
       NvU32 levelId;     // level indicator for a policy, such
                            as: NVAPI_PERF_LEVEL_CONSERVATIVE
       NvU32 currentLevel; // new cooler level for the selected
                             policy level indicator.
      NvU32 defaultLevel; // default cooler level for the
                             selected policy level indicator.
     } policyCoolerLevel[NVAPI_MAX_LEVELS_PER_POLICY];
 } NV_GPU_COOLER_POLICY_TABLE;
```

NvAPI_GPU_GetCoolerSettings()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.60

NV_GPU_GETCOOLER_SETTINGS Structure: Version 1

This function retrieves the cooler information for all coolers or for a specific cooler associated with the selected GPU.

Coolers are indexed 0 to NVAPI_MAX_COOLERS_PER_GPU - 1.

- To retrieve specific cooler info set the **`coolerIndex`** to the appropriate cooler index.
- To retrieve info for all coolers set **`coolerIndex`** to `NVAPI_COOLER_TARGET_ALL`.

Use **`NV_GPU_GETCOOLER_SETTINGS_VER`** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetCoolerSettings(
                NvPhysicalGpuHandle        hPhysicalGpu,
                NvU32                      coolerIndex,
                NV_GPU_GETCOOLER_SETTINGS *pCoolerInfo);
```

Input Parameter

`hPhysicalGpu`	Handle for the physical GPU
`coolerIndex`	Explicit cooler index selection

Output Parameter

`pCoolerInfo`	Pointer to the array of cooler settings. See “NV_GPU_GETCOOLER_SETTINGS Structure” on page 103.

Return Status[i]

`NVAPI_OK`	Request completed
`NVAPI_ERROR`	Miscellaneous error occurred.
`NVAPI_INVALID_ARGUMENT`	pCoolerInfo is NULL
`NVAPI_HANDLE_INVALIDATED`	Handle passed has been invalidated
`NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE`	Handle passed is not a physical GPU handle
`NVAPI_INCOMPATIBLE_STRUCT_VERSION`	The version of the INFO struct is not supported

i. See “NvAPI Return Status Codes” on page 13 for a list of other possible return codes.

NvAPI_GPU_SetCoolerLevels()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.60

NV_GPU_GETCOOLER_LEVEL Structure: Version 1

This function sets the cooler levels for all coolers or for a specific cooler associated with the selected GPU.

Coolers are indexed 0 to NVAPI_MAX_COOLERS_PER_GPU - 1. Every cooler level with non-zero current policy gets applied.

The new level should be in the range of minlevel and maxlevel retrieved from GetCoolerSettings API or between NVAPI_MIN_COOLER_LEVEL and MAX_COOLER_LEVEL.

- ❑ To set level for a specific cooler set the **`coolerIndex`** to the appropriate cooler index.
- ❑ To set level for all coolers set **`coolerIndex`** to `NVAPI_COOLER_TARGET_ALL`.

Note: To lock the fan speed independent of the temperature or performance changes, set the cooler currentPolicy to NVAPI_COOLER_POLICY_MANUAL else set it to the current policy retrieved from the GetCoolerSettings API.

Use **`NV_GPU_GETCOOLER_LEVEL_VER`** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_SetCoolerLevels(
                NvPhysicalGpuHandle     hPhysicalGpu,
                NvU32                   coolerIndex,
                NV_GPU_SETCOOLER_LEVEL *pCoolerLevels);
```

Input Parameter

`hPhysicalGpu`	Handle to the physical GPU
`coolerIndex`	Explicit cooler index selection
`pCoolerLevels`	Updated cooler level and cooler policy See “NV_GPU_SETCOOLER_LEVEL Structure” on page 104.

Return Status[i]

`NVAPI_OK`	Request completed.
`NVAPI_ERROR`	Miscellaneous error occurred.
`NVAPI_INVALID_ARGUMENT`	pCoolerLevels is NULL.
`NVAPI_HANDLE_INVALIDATED`	The handle passed has been invalidated.

Return Status[i]

`NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE`	The handle passed is not a physical GPU handle.
`NVAPI_INCOMPATIBLE_STRUCT_VERSION`	The version of the INFO struct is not supported.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_RestoreCoolerSettings()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.60

This function restores the modified cooler settings to the NVIDIA default settings.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_RestoreCoolerSettings(
                NvPhysicalGpuHandle  hPhysicalGpu,
                NvU32               *pCoolerIndex,
                NvU32                coolerCount);
```

Input Parameter

`hPhysicalGPU`	Handle for the physical GPU
`pCoolerIndex`	Pointer to the array containing absolute cooler indexes to restore Pass NULL to restore all coolers.
`coolerCount`	Number of coolers to restore

Return Status[i]

`NVAPI_OK`	Request completed.
`NVAPI_ERROR`	Miscellaneous error occurred.
`NVAPI_HANDLE_INVALIDATED`	The handle passed has been invalidated.
`NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE`	The handle passed is not a physical GPU handle.
`NVAPI_INCOMPATIBLE_STRUCT_VERSION`	The version of the INFO struct is not supported.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetCoolerPolicyTable()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.60

NV_GPU_COOLER_POLICY_TABLE Structure: Version 1

This function retrieves the table of cooler and policy levels for the selected policy. Supported only for NVAPI_COOLER_POLICY_PERF.

Use **NV_GPU_COOLER_POLICY_TABLE_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetCoolerPolicyTable(
                NvPhysicalGpuHandle          hPhysicalGpu,
                NvU32                        coolerIndex,
                NV_GPU_COOLER_POLICY_TABLE *pCoolerTable,
                NvU32                       *count);
```

Input Parameter

hPhysicalGPU	Handle for the physical GPU
coolerIndex	Cooler index selection

Output Parameter

pCoolerTable	Pointer to the table of policy levels and associated cooler levels See "NV_GPU_COOLER_POLICY_TABLE Structure" on page 104.
count	Count of the number of valid levels for the selected policy.

Return Status[i]

NVAPI_OK	Request completed.
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_HANDLE_INVALIDATED	The handle passed has been invalidated.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	The handle passed is not a physical GPU handle.
NVAPI_INCOMPATIBLE_STRUCT_VERSION	The version of the INFO struct is not supported.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_SetCoolerPolicyTable()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.60

NV_GPU_COOLER_POLICY_TABLE Structure: Version 1

This function restores the modified cooler settings to NVIDIA defaults. Supported only for NVAPI_COOLER_POLICY_PERF.

Use **NV_GPU_COOLER_POLICY_TABLE_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_SetCoolerPolicyTable(
                NvPhysicalGpuHandle            hPhysicalGpu,
                NvU32                          coolerIndex,
                NV_GPU_COOLER_POLICY_TABLE    *pCoolerTable,
                NvU32                          count);
```

Input Parameter

hPhysicalGPU	Handle for the physical GPU
coolerIndex	Cooler index selection
pCoolerTable	Updated table of policy levels and associated cooler levels. See "NV_GPU_COOLER_POLICY_TABLE Structure" on page 104. Every non-zero policy level gets updated.
count	Number of valid levels in the policy table

Return Status[i]

NVAPI_OK	Request completed
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_HANDLE_INVALIDATED	The handle passed has been invalidated.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	The handle passed is not a physical GPU handle.
NVAPI_INCOMPATIBLE_STRUCT_VERSION	The version of the INFO struct is not supported.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_RestoreCoolerPolicyTable()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.60

This function restores the perf table policy levels to the defaults.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_RestoreCoolerPolicyTable(
                NvPhysicalGpuHandle  hPhysicalGpu,
                NvU32               *pCoolerIndex,
                NvU32                coolerCount,
                NV_COOLER_POLICY     policy);
```

Input Parametesr

hPhysicalGPU	Handle for the physical GPU
coolerIndex	The cooler index selection.
pCoolerIndex	Array containing absolute cooler indexes to restore. Pass NULL restore all coolers.
coolerCount	Number of coolers to restore
policy	Restore for the selected policy See "NV_COOLER_POLICY Enum" on page 102.

Return Status[i]

NVAPI_OK	Request completed
NVAPI_ERROR	Miscellaneous error occurred.
NVAPI_HANDLE_INVALIDATED	The handle passed has been invalidated.
NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE	The handle passed is not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

Thermal API Calls

The calls in this section get temperature levels from various thermal sensors associated with the GPU.

- "NvAPI_GPU_GetThermalSettings())" on page 112

Thermal API Defines and Structures

```
#define NVAPI_MAX_THERMAL_SENSORS_PER_GPU 3
```

THERMAL_TARGET Enum

```
typedef enum
{
    NVAPI_THERMAL_TARGET_NONE           = 0,
    NVAPI_THERMAL_TARGET_GPU            = 1,
    NVAPI_THERMAL_TARGET_MEMORY         = 2,
    NVAPI_THERMAL_TARGET_POWER_SUPPLY   = 4,
    NVAPI_THERMAL_TARGET_BOARD          = 8,
    NVAPI_THERMAL_TARGET_ALL            = 15,
    NVAPI_THERMAL_TARGET_UNKNOWN        = -1,
} NV_THERMAL_TARGET;
```

THERMAL_CONTROLLER Enum

```
typedef enum
{
    NVAPI_THERMAL_CONTROLLER_NONE = 0,
    NVAPI_THERMAL_CONTROLLER_GPU_INTERNAL,
    NVAPI_THERMAL_CONTROLLER_ADM1032,
    NVAPI_THERMAL_CONTROLLER_MAX6649,
    NVAPI_THERMAL_CONTROLLER_MAX1617,
    NVAPI_THERMAL_CONTROLLER_LM99,
    NVAPI_THERMAL_CONTROLLER_LM89,
    NVAPI_THERMAL_CONTROLLER_LM64,
    NVAPI_THERMAL_CONTROLLER_ADT7473,
    NVAPI_THERMAL_CONTROLLER_SBMAX6649,
    NVAPI_THERMAL_CONTROLLER_VBIOSEVT,
    NVAPI_THERMAL_CONTROLLER_OS,
    NVAPI_THERMAL_CONTROLLER_UNKNOWN = -1,
} NV_THERMAL_CONTROLLER;
```

NV_GPU_THERMAL_SETTINGS Structure

```
typedef struct
{
    NvU32   version;   //Structure version
    NvU32   count;     //Number of thermal sensors associated
                         with the selected GPU
    struct
    {
       NV_THERMAL_CONTROLLER   controller;  //
                     See “THERMAL_CONTROLLER Enum” on page 111.
       NvU32     defaultMinTemp;  //The minimum default
                                temperature value of the
                                thermal sensor in degrees
                                centigrade
       NvU32     defaultMaxTemp;  //The maximum default
                                temperature value of the
                                thermal sensor in degrees
                                centigrade
       NvU32     currentTemp;  //The current temperature
                                value of the thermal sensor
                                in degrees centigrade
       NV_THERMAL_TARGET      target; //Thermal sensor target
                          See “THERMAL_TARGET Enum” on page 111.
    } sensor[NVAPI_MAX_THERMAL_SENSORS_PER_GPU];

} NV_GPU_THERMAL_SETTINGS;
```

NvAPI_GPU_GetThermalSettings())

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 87.70

NV_GPU_THERMAL_SETTINGS Structure: Version 1

This function retrieves the thermal information of all thermal sensors or specific thermal sensor associated with the selected GPU.

Thermal sensors are indexed 0 to `NVAPI_MAX_THERMAL_SENSORS_PER_GPU` - 1.

- To retrieve specific thermal sensor information, set the **`sensorIndex`** to the required thermal sensor index.
- To retrieve information for all sensors, set **`sensorIndex`** to `NVAPI_THERMAL_TARGET_ALL`.

Use **`NV_GPU_THERMAL_SETTINGS_VER`** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetThermalSettings(
                NvPhysicalGpuHandle      hPhysicalGpu,
                NvU32                    sensorIndex,
                NV_GPU_THERMAL_SETTINGS *pThermalSettings);
```

Input Parameter

`hPhysicalGPU`	Handle for the physical GPU
`sensorIndex`	Explicit thermal sensor index selection

Output Parameter

`pThermalSettings`	Pointer to the thermal settings. See NV_GPU_THERMAL_SETTINGS Structure.

Return Status[i]

`NVAPI_OK`	Request completed
`NVAPI_ERROR`	Miscellaneous error occurred.
`NVAPI_HANDLE_INVALIDATED`	The handle passed has been invalidated.
`NVAPI_INVALID_ARGUMENT`	**`pThermalInfo`** is NULL
`NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE`	The handle passed is not a physical GPU handle.
`NVAPI_INCOMPATIBLE_STRUCT_VERSION`	The version of the INFO struct is not supported.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

GPU Topology Configuration Calls

The APIs in this section provide the ability to define one or more SLI devices and standalone GPU topology.

- "NvAPI_GetValidGpuTopologies()" on page 116
- "NvAPI_SetGpuTopologies()" on page 117
- "NvAPI_GPU_GetPerGpuTopologyStatus" on page 118
- "NvAPI_GPU_GetAllGpusOnSameBoard" on page 119

GPU Topology Enums and Structures

NV_GPU_TOPOLOGY_STATUS_FLAGS Enum

```
typedef enum
{
 NV_GPU_TOPOLOGY_STATUS_OK                            = 0x00000000,
            //SLI is capable, and the topology "status" field
               indicates this state.
  NV_GPU_TOPOLOGY_STATUS_INVALID_GPU_COUNT             = 0x00000001,
            //SLI is NOT capable, and the "pStatus" parameter in
              NvAPI_GetValidGpuTopologies indicates these states.
NV_GPU_TOPOLOGY_STATUS_OS_NOT_SUPPORTED              = 0x00000002,
 NV_GPU_TOPOLOGY_STATUS_OS_ERROR                       = 0x00000004,
 NV_GPU_TOPOLOGY_STATUS_NO_VIDLINK                     = 0x00000008,
 NV_GPU_TOPOLOGY_STATUS_INSUFFICIENT_LINK_WIDTH        = 0x00000010,
NV_GPU_TOPOLOGY_STATUS_CPU_NOT_SUPPORTED              = 0x00000020,
NV_GPU_TOPOLOGY_STATUS_GPU_NOT_SUPPORTED              = 0x00000040,
NV_GPU_TOPOLOGY_STATUS_BUS_NOT_SUPPORTED              = 0x00000080,
NV_GPU_TOPOLOGY_STATUS_NON_APPROVED_CHIPSET           = 0x00000100,
NV_GPU_TOPOLOGY_STATUS_VBIOS_NOT_SUPPORTED            = 0x00000200,
 NV_GPU_TOPOLOGY_STATUS_GPU_MISMATCH                    = 0x00000400,
NV_GPU_TOPOLOGY_STATUS_ARCH_MISMATCH                  = 0x00000800,
NV_GPU_TOPOLOGY_STATUS_IMPL_MISMATCH                  = 0x00001000,
NV_GPU_TOPOLOGY_STATUS_REV_MISMATCH                   = 0x00002000,
 NV_GPU_TOPOLOGY_STATUS_NON_PCIE_BUS                    = 0x00004000,
NV_GPU_TOPOLOGY_STATUS_FB_MISMATCH                    = 0x00008000,
NV_GPU_TOPOLOGY_STATUS_VBIOS_MISMATCH                 = 0x00010000,
NV_GPU_TOPOLOGY_STATUS_QUADRO_MISMATCH                = 0x00020000,
NV_GPU_TOPOLOGY_STATUS_BUS_TOPOLOGY_ERROR             = 0x00040000,
} NV_GPU_TOPOLOGY_STATUS_FLAGS;
```

NV_SET_GPU_TOPOLOGY_FLAGS Enum

```
typedef enum
{
    NV_SET_GPU_TOPOLOGY_DEFER_APPLY              = 0x00000001,
                          //Calling application controls
                            the reload of the display driver
    NV_SET_GPU_TOPOLOGY_DEFER_3D_APP_SHUTDOWN    = 0x00000002,
                          //Calling application will force
                            shutdown of 3D applications that
                            hold hardware resources.
    NV_SET_GPU_TOPOLOGY_DEFER_DISPLAY_RECONFIG   = 0x00000004,
                          //Calling application will control
                            display configuration required
                            for SetGpuTopology() to work.
} NV_SET_GPU_TOPOLOGY_FLAGS;
```

NV_GPU_TOPOLOGY _FLAGS

```
typedef enum
{
  NV_GPU_TOPOLOGY_ACTIVE                        = 0x00000001,
         //Topology "flags" field to indicate if SLI is ACTIVE
          on this topology of GPUs. Read only.
   NV_GPU_TOPOLOGY_VIDLINK_PRESENT               = 0x00000002,
         //Topology "flags" field to indicate if Video link is
           present.
  NV_GPU_TOPOLOGY_COMPUTE                       = 0x00010000,
         //Reserve the GPUs in this topology for compute.
  NV_GPU_TOPOLOGY_SLIMULTIMON                   = 0x00020000,
         //This topology allows multi-display SLI output.
} NV_GPU_TOPOLOGY_FLAGS;
```

NV_GPU_TOPOLOGY Structure

This structure defines a set of all GPUs present in a system.

All GPUs with the same parentNdx value describe a single logical GPU.

GPUs that have a unique parentNdx represent standalone GPUs.

The values returned in parentNdx are arbitrary. They are only used to determine which physical GPUs belong to the same logical GPU.

```
typedef struct
{
  NvU32               version;  //Structure version
  NvU32               gpuCount; //Count of GPUs in this topology
```

NV_GPU_TOPOLOGY Structure

```
  NvPhysicalGpuHandle hPhysicalGpu[NVAPI_MAX_GPU_PER_TOPOLOGY];
                               //Array of GPU handles
  NvU32               displayGpuIndex;
                               //Index of the display GPU owner
                                 in the GPU array
  NvU32               displayOutputTargetMask;
                               //Target display device mask
  NvU32               flags;   //One or more topology flags from
                                 NV_GPU_TOPOLOGY_FLAGS
  NvU32               status;  //One of the flags in
                                 NV_GPU_TOPOLOGY_STATUS_FLAGS
} NV_GPU_TOPOLOGY;
```

NV_GPU_VALID_GPU_TOPOLOGIES Structure

```
typedef struct
{
   NvU32           version;         //Structure version
   NvU32            gpuTopoCount;    //Count of valid topologies
   NV_GPU_TOPOLOGY   gpuTopo[NVAPI_MAX_AVAILABLE_GPU_TOPOLOGIES];
                                    //Maximum gputopologies
} NV_GPU_VALID_GPU_TOPOLOGIES;
```

NvAPI_GetValidGpuTopologies()

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version: 97.10

NV_GPU_VALID_GPU_TOPOLOGIES Structure: Version 1

This function returns all valid GPU topologies that can be used to configure the physical GPUs using **`NvAPI_SetGpuTopologies()`**.

The call also returns the current active topologies in an array of `NV_GPU_TOPOLOGY` structs; one for each valid configuration of GPUs present in the system. This generated list is valid as long as GPUs remain in the same slots in the system. It is not affected by which GPUs are presently in use.

NV_GPU_TOPOLOGY.displayGpuIndex returned will match the boot GPU if it exists as an active topology. If it not an active topology, it points to the "first" GPU that has a display monitor connected.

Use **NV_GPU_VALID_GPU_TOPOLOGIES_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GetValidGpuTopologies(
                NV_GPU_VALID_GPU_TOPOLOGIES *pTopology,
                NvU32                       *pstatus);
```

Output Parameter

ptopology	An array of *pCount topology structures. See "NV_GPU_VALID_GPU_TOPOLOGIES Structure" on page 116 Use NvAPI_SetGpuTopologies() to set up one of these GPU topologies.
pStatus	Any system status returned in case zero topology is retrieved. System status is one or more flags in NV_GPU_TOPOLOGY_STATUS_FLAGS Enum when SLI is NOT capable.

Return Status[i]

NVAPI_OK	Call succeeded; One or more GPU topologies were returned
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.
NVAPI_INVALID_ARGUMENT	One or more arguments are invalid.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_SetGpuTopologies()

OS/architecture : Windows XP / 32-bit and 64-bit,
Earliest ForceWare Version: 97.10
NV_GPU_VALID_GPU_TOPOLOGIES Structure: Version 1

This function configures the physical GPUs in the system into one or more logical devices defined by the **NV_GPU_TOPOLOGY** structure.

It is recommended that the caller application do the following:

- Save the current GPU topology retrieved from the APIs **NvAPI_EnumLogicalGPUs()** and **NvAPI_GetPhysicalGPUsFromLogicalGPU()**.
- Save the current view state for associated displays on these GPUs using the NvAPI_GetView() .
- Set **NV_GPU_TOPOLOGY.displayGpuIndex** to the GPU index in the topology with an active display connection.

- If **DEFER_3D_APP_SHUTDOWN** is not set, then notify the user that all 3D applications will be forced to close.
- Do not create 3D handles or objects that can block the topology transition.

Use **NV_GPU_VALID_GPU_TOPOLOGIES_VER** to initialize the structure version.

Function Prototype

```
NVAPI_INTERFACE NvAPI_SetGpuTopologies(
                NV_GPU_VALID_GPU_TOPOLOGIES *pTopology,
                NvU32                        flags);
```

Input Parameter

pTopology	Pointer to an array of structures defining the desired GPU topology retrieved from NvAPI_GetValidGpuTopologies(). See "NV_GPU_VALID_GPU_TOPOLOGIES Structure" on page 116.
flags	See "NV_SET_GPU_TOPOLOGY_FLAGS Enum" on page 115.

Return Status[i]

NVAPI_OK	The call succeeded.
NVAPI_ERROR	Check the status returned in pTopology->status.
NVAPI_INVALID_ARGUMENT	One or more arguments are invalid.
NVAPI_NVIDIA_DEVICE_NOT_FOUND	No NVIDIA GPU driving a display was found.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetPerGpuTopologyStatus

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit
Earliest ForceWare Version:

This function returns per-GPU topology state flags from NV_GPU_TOPOLOGY_STATUS_FLAGS for the queried GPU handle.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetPerGpuTopologyStatus(
                NvPhysicalGpuHandle  hPhysicalGpu,
                NvU32               *pStatus);
```

Input Parameter

hPhysicalGPU	Handle for the selected GPU

Output Parameter

pStatus	Indicates one or more flags from NV_GPU_TOPOLOGY_STATUS_FLAGS Enum, which are the subset of the same flags retrieved from NV_GPU_TOPOLOGY.status or pStatus in NvAPI_GetValidGpuTopologies(). **Note**: The per-GPU topology status can be queried whether the queried GPU is part of a topology or not.

Return Status[i]

`NVAPI_OK`	Request completed
`NVAPI_ERROR`	Miscellaneous error
`NVAPI_HANDLE_INVALIDATED`	Handle passed has been invalidated.
`NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE`	Handle passed is not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.

NvAPI_GPU_GetAllGpusOnSameBoard

OS/architecture : Windows XP / 32-bit and 64-bit,
Windows Vista / 32-bit and 64-bit

Earliest ForceWare Version:

This function returns a set of GPUs handles that exists on the same board as the queried GPU handle.

Function Prototype

```
NVAPI_INTERFACE NvAPI_GPU_GetAllGpusOnSameBoard(
            NvPhysicalGpuHandle hPhysicalGpu,
           NvPhysicalGpuHandle nvGPUHandle[NVAPI_MAX_PHYSICAL_GPUS],
            NvU32                *pGpuCount);
```

Input Parameter

`hPhysicalGPU`	Handle for the queried GPU

Output Parameter

nvGPUHandle	The associated GPUs on the same board as the queried GPU. This array includes the queried GPU handle.
`pGpuCount`	Pointer to the count of GPUs that exists on the same board. This count includes the queried GPU handle.

Return Status[i]

`NVAPI_OK`	Request completed
`NVAPI_ERROR`	Miscellaneous error
`NVAPI_HANDLE_INVALIDATED`	Handle passed has been invalidated.
`NVAPI_EXPECTED_PHYSICAL_GPU_HANDLE`	Handle passed is not a physical GPU handle.

i. See "NvAPI Return Status Codes" on page 13 for a list of other possible return codes.