QUADRO SDI CAPTURE

PG-04817-001_v05 | May 2011

Programmer’s Guide

DOCUMENT CHANGE HISTORY

PG-04817-001_v05

Version	Date	Authors	Description of Change
01	August 11, 2009	TT, SM	Initial Release
02	January 14, 2010	TT, SM	•Updated the following chapters: 5, 8 and 9 •Added Chapter 13 “Ancillary Data API”
03	June 18, 2010	TT, SM	•Changes to Section 4.6 •Added note to Section 5.3 •Updated code in Section 8.2 •Added Section 8.4 “Audio” •Updated Chapter 9 •Updated codes in Chapters 11, 12, and 13
04	November 8, 2010	TT, SM	•Updated the following sections: 4.3, 4.6, and 5.1 •Updated Code Listing 23 in Section 8.4.3 •Updated Code Listing 29 in Section 9.2 •Added Section 6.1 “Typical Color Space Conversion” •Added Section 9.3 “Multiple Capture Cards”
05	May 26, 2011	TT, SM	•Minor text edits •Updated Code Listing 8 and Code Listing 12

TABLE OF CONTENTS

1 GETTING STARTED

The NVIDIA Quadro® SDI Capture is a PCI Express ×8 interface card capable of capturing up to four single-link, or two dual-link HD SDI, or two 3G SDI video streams to onboard graphics processing unit (GPU) memory or system memory. This document describes programming methodologies for the NVIDIA® Quadro® SDI Capture board on systems running either the Windows or Linux operating systems.

Application programming of the Quadro SDI Capture is broken into two principle parts, device control and data transfer. Device control handles the hardware device configuration as well as the starting and stopping of data transfers while data transfer is the sequence of operations that transfers incoming video data to either GPU memory or system memory. Incoming 8, 10 or 12-bit SDI video data is captured in GPU memory as either one or more OpenGL Video Buffer Objects (VBO) or Texture Objects according to the data transfer parameters set by the application. These VBOs or textures are then available for further processing by the Quadro GPU. Meanwhile, the associated ancillary data is transferred into buffers in system memory for further application processing.

The remainder of this document will outline the general procedures required to program the Quadro SDI Capture device on systems running either the Windows or Linux operating systems. Some examples may refer to the companion NVIDIA Quadro SDI Output. For more complete information on programming that device, see the *Quadro SDI Output Programming Guide*.

2 DEVICE CONTROL APIS

2.1 WINDOWS

On systems running the Microsoft Windows Operating System, hardware setup and control is handled by the VIO commands of NVAPI, NVIDIA's universal device control API. Use of NVAPI requires the inclusion of the following include and library files. These files are packaged within the NVIDIA SDI Video SDK.

```
nvapi.h
```

```
nvapi.lib
```

Use of the NVAPI to control the Quadro SDI capture device is described in Chapter 4 *Device Setup and Control*. For additional information on NVAPI, refer to the NVAPI Online Help documentation.

2.2 LINUX

On a Linux-based system, hardware setup and control is enabled by the NV-CONTROL X extension. Use of the NV-CONTROL X extension requires the following include files. These files are packaged within the nvidia-settings-1.0.tar.gz archive that is included with the NVIDIA SDI video or display driver.

```
NVCtrlLib.h
```

```
NVCtrl.h
```

Control of the Quadro SDI Capture device with the NV-CONTROL X Extension is described in Chapter 4 *Device Setup and Control*. Additional information on the NV-CONTROL X Extension can be found in the NV-CONTROL-API.txt document included in the archive listed.

3 OPENGL API

Transfer of the SDI video data is enabled by an extension to OpenGL. The `GL_NV_video_capture` extension provides a mechanism for the direct capture and streaming of the incoming SDI video into either video buffer objects (VBO) or texture objects in GPU memory. Data captured into VBOs may be further processed on the GPU using either OpenCL or NVIDIA®'s CUDA™ or transferred to system memory by the application. Texture objects can be immediately used for GPU rendering as OpenGL texture maps. This OpenGL extension is available on both Windows and Linux systems when the Quadro SDI Capture device is installed into the system and the device driver software installed. Use of this extension will be demonstrated in Chapter 5. Procedures and tokens for this extension are listed in Chapter 10. For additional detail on this specification, see the `GL_NV_video_capture` OpenGL specification.

In addition to the above OpenGL extensions, other useful extensions include the following:

- NV_present_video
- ARB_vertex_buffer_object
- EXT_timer_query
- EXT_framebuffer_object
- NV_gpu_affinity

Additional information on these OpenGL extensions can be found in the extension specifications located at http://developers.nvidia.com or http://www.opengl.org/.

4 DEVICE SETUP AND CONTROL

The Quadro SDI Capture device must be properly configured for the input video signal format, data format, color space, etc. before input video can be captured. This hardware configuration is performed by NVAPI on Microsoft Windows-based systems and the NV-CONTROL X extension on Linux-based systems. The remainder of this section will describe the step by step process required to configure the video device.

4.1 DEFINITIONS

Before getting started, it is important to understand terminology used in the configuration process. Here is a list of terms and definitions useful in configuring the Quadro SDI Capture card.

- **Jack**
  The term **jack** refers to the physical I/O connector on the Quadro SDI capture device. A single Quadro SDI capture device has four input jacks. A jack has the capability to carry one or more video payloads or channels. Multiple capture cards can be utilized within a system to provide additional video input channels.

- **Channel**
  A **channel** describes a video payload available on a jack. Jacks have the ability to carry multiple channels. The current limitation of the Quadro SDI Capture device is a maximum of two channels per jack.

- **Stream**
  A combination of channels across one or more jacks form a **stream**. The control API, either NVAPI on Windows or NV CONTROL on Linux is utilized by an application to combine the available active video payload channels into streams for capture. Chapter 4 describes the methodologies for specifying streams within an application. The Quadro SDI capture device captures streams to video buffer or texture objects. Chapter 5 outlines how video buffers and / or texture objects are bound to streams for capture.

4.2 NVAPI INITIALIZATION

On Windows systems, prior to using NVAPI to configure the video capture device, NVAPI must be initialized by calling `NVAPI_Initialize()` as shown in the following code sample.

Code Listing 1: NVAPI Initialization

```
// Initialize NVAPI
if (NvAPI_Initialize() != NVAPI_OK) {
      MessageBox(NULL, "Error Initializing NVAPI.",
      "Error", MB_OK);
      return E_FAIL;
}
```

4.3 LOCATING VIDEO CAPTURE DEVICE

Prior to configuring the video capture device, query the available video I/O topologies to find available video capture devices. On Windows, this is done by examining the device capabilities using NVAPI to confirm that the adapter capabilities include `NVVIOCAPS_VIDIN_SDI`. Once the desired video device is found, the application should save the VIO handle and the VIO ID. VIO ID will be sued later in the OpenGL setup state to make sure that OpenGL and NVAPI are utilizing the same capture device if there are multiple devices in the system. The procedure for doing this is outlined in Code Listing 2.

Code Listing 2: Using NVAPI to Query Video I/O Topologies to Find Video Capture Device on Windows

```
// Query Available Video I/O Topologies
memset(&l_vioTopos, 0, sizeof(l_vioTopos));
l_vioTopos.version = NVVIOTOPOLOGY_VER;
if (NvAPI_VIO_QueryTopology(&l_vioTopos) != NVAPI_OK) {
      MessageBox(NULL, "Video I/O Unsupported.", "Error", MB_OK);
      return E_FAIL;
}

// Cycle through all SDI topologies looking for the first
// available SDI input device.
l_bFound = FALSE;
i = 0;
while ((i < l_vioTopos.vioTotalDeviceCount) && (!l_bFound)) {

      // Get video I/O capabilities for current video I/O target.
      memset(&l_vioCaps, 0, sizeof(l_vioCaps));
      l_vioCaps.version = NVVIOCAPS_VER;
      if (NvAPI_VIO_GetCapabilities(l_vioTopos.vioTarget[i].hVioHandle,
                                    &l_vioCaps) != NVAPI_OK) {

            MessageBox(NULL, "Video I/O Unsupported.", "Error", MB_OK);
            return E_FAIL;
      }

      // If video input device found, set flag.
      if (l_vioCaps.adapterCaps & NVVIOCAPS_VIDIN_SDI) {
            m_vioHandle = l_vioTopos.vioTarget[i].hVioHandle;
            m_vioID = 1_vioTopos.vioTarget[i] vioID;
            l_bFound = TRUE;
      } else {
            i++;
      }
} // while i < vioTotalDeviceCount

// Video input device found, save VIO handle and the VIO ID.
// Otherwise, if no video input device found, return error.
if (!l_bFound) {
      MessageBox(NULL, "No SDI video input devices found.", "Error",
                 MB_OK);
      return E_FAIL;
}
```

On Linux, use the `XNVCTRLQueryTargetCount()` function to query the number of video capture devices available in the system. This call will fail if there are no video capture devices in the system. In case of success, the application should save the GVI handle to make sure that the OpenGL stage is configuring the same capture device as the NVCtrl stage (for the case when multiple capture devices are present in the system).

Code Listing 3: Using NV_CONTROL-X Extension to Locate Available Video Capture Devices on Linux

```
// Query number of SDI video capture devices in the system.
if (!XNVCTRLQueryTargetCount(dpy, NV_CTRL_TARGET_TYPE_GVI,
&numVideoIn)) {
    fprintf(stderr, "No video capture devices avaiable.\n");
    return GL_FALSE;
}
// Select the first available capture device and query its identifier
XNVCTRLQueryTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI,
                            0, 0, NV_CTRL_GVI_GLOBAL_IDENTIFIER,
                            &hGVI))
```

4.4 DETECTING INPUT STATUS

At this point, prior to configuring the video capture device, an application may want to query the video input signals detected for each channel at each of the jacks. On Windows, this is done with the NVAPI `NvAPI_VIO_Status()` function as demonstrated in the following Code Listing 4.

Code Listing 4: Query Input Status on Windows

```
// Get status of all jacks on the input device
memset(&l_hVioStatus, 0, sizeof(l_hVioStatus));
l_hVioStatus.version = NVVIOSTATUS_VER;
ret = NvAPI_VIO_Status(m_vioHandle, &l_hVioStatus);
//if (NvAPI_VIO_Status(m_vioHandle, &l_hVioStatus) != NVAPI_OK) {
if (ret != NVAPI_OK) {
      MessageBox(NULL, "Cannot get status of SDI input device.",
"Error", MB_OK);
      return E_FAIL;
}

// Cycle through the jacks and display the status of
// each active channel.
for (i=0; i < NVAPI_MAX_VIO_JACKS; i++) {
      for (j=0; j < NVAPI_MAX_VIO_CHANNELS_PER_JACK; j++) {

      DumpChannelStatus(l_hVioStatus.vioStatus.inStatus.vidIn[i][j]);
      }
}
```

On Linux, the NV Control function `XNVCTRLQueryTargetAttribute()` can be utilized to query the state of input jacks.

Code Listing 5: Query Input Status on Linux

```
// Query the number of active jacks on the SDI video capture device
// For now, simply query the first video capture device found.
if (!XNVCTRLQueryTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI,
                                 0, 0, NV_CTRL_GVI_NUM_PORTS,
                                 &numJacks)){
  fprintf(stderr, "Unable to query active jacks on video capture
device.\n");
  return GL_FALSE;
}

// Print number of active video jacks found on the capture device
fprintf(stderr, "Number of active jacks: %d\n", numPorts);

// Query and print video input signal information
// detected at each jack of the video input device.
for (int i = 0; i < numJacks; i++) {

  // Signal format.
  if (!XNVCTRLQueryTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI,
                        0, i, NV_CTRL_GVIO_DETECTED_VIDEO_FORMAT,
                        &value)) {
    fprintf(stderr, "Jack %d : Cannot detect input video format\n", i);
  } else {
    fprintf(stderr, "Jack %d : Video format: %s\n", i,
            decodeSignalFormat(value));
  }

  // Bits per color component.
  if (!XNVCTRLQueryTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI,
              0, i, NV_CTRL_GVI_DETECTED_PORT_BITS_PER_COMPONENT,
              &value)) {
    fprintf(stderr, "Jack %d : Cannot detect bits per component\n", i);
  } else {
    fprintf(stderr, "Jack %d : Bits per component: %s\n", i,
            decodeBitsPerComponent(value));
  }

  // Component sampling.
  if (!XNVCTRLQueryTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI,
              0, i, NV_CTRL_GVI_DETECTED_PORT_COMPONENT_SAMPLING,
              &value)) {
    fprintf(stderr, "Jack %d : Cannot detect sampling\n");
  } else {
    fprintf(stderr, "Jack %d : Sampling:: %s\n", i,
            decodeComponentSampling(value));
  }

  // Color space.
  if (!XNVCTRLQueryTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI,
            0, i, NV_CTRL_GVI_DETECTED_PORT_COLOR_SPACE, &value)) {
    fprintf(stderr, "Jack %d : Cannot detect color space.\n", i);
  } else {
    fprintf(stderr, "Jack %d : Color spsce: %s\n", i,
            decodeColorSpace(value));
  }


  // Link ID.
  if (!XNVCTRLQueryTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI,
                             0, i, NV_CTRL_GVI_DETECTED_PORT_LINK_ID,
                             &value)) {
    fprintf(stderr, "Jack %d : Cannot detect link ID.\n", i);
  } else {
    fprintf(stderr, "Jack %d : Link ID: %d\n", i, value);
  }
} // for numPorts
```

4.5 QUERYING STREAM CONFIGURATION

Active channels on jacks are configured into streams. These streams are subsequently captured by the video capture device. It may also be useful for an application to query the stream configuration. The following code example demonstrates how this is done with NVAPI.

Code Listing 6: Querying the Stream Configuration

```
// Get the stream configuration of the input device
memset(&l_vioConfig, 0, sizeof(l_vioConfig));
l_vioConfig.version = NVVIOCONFIG_VER;
l_vioConfig.nvvioConfigType = NVVIOCONFIGTYPE_IN;
l_vioConfig.fields = NVVIOCONFIG_SIGNALFORMAT | NVVIOCONFIG_STREAMS;
if (NvAPI_VIO_GetConfig(m_vioHandle, &l_vioConfig) != NVAPI_OK) {
      MessageBox(NULL, "Cannot get configuration of SDI input device.",
"Error", MB_OK);
      return E_FAIL;
}

// Display stream configuration of input device.
fprintf(stderr, "\nNumber of Streams: %d\n",
l_vioConfig.vioConfig.inConfig.numStreams);
fprintf(stderr, "Signal Format: %s\n",
DecodeSignalFormat(l_vioConfig.vioConfig.inConfig.signalFormat));

// Display the configuration of each stream.
for (i=0; i < l_vioConfig.vioConfig.inConfig.numStreams; i++) {
      DumpStreamStatus(l_vioConfig.vioConfig.inConfig.streams[i]);
}
```

4.6 CONFIGURING THE VIDEO CAPTURE DEVICE

The video capture device must be configured for the desired video signal format, the number of streams to capture and the number of surfaces in the ring buffer. The amount of surfaces in the ring buffer is set by specifying the `numRawCaptureImages` field in the vioConfig structure in NvAPI and by assigning an `NV_CTRL_GVI_NUM_CAPTURE_SURFACES` attribute value in `NVCtrl`.

A lower number of capture surfaces will mean less video memory is used, but can result in frames being dropped if the application cannot keep up with the capture device. A higher number will prevent frames from being dropped making capture more reliable but will consume more of pinned video memory.

Note: In WDDM graphics driver model the number of capture surfaces is limited by the maximum allowed amount of allocated pinned video memory which is 25% of the total video memory. It is important to remember that each capture surface is always big enough to contain 4 video streams of video, plus ancillary data. This imposes much stricter limits on the ring buffer size than the NvAPI/NVCtrl allowed maximum which is normally set to 32.

Subsequently, each stream to be captured must be configured. Stream configuration consists of setting the component bit sampling (8, 10, or 12-bit), 422 to 444 component expansion, the number of links and the corresponding jacks and channels. An example of how this is done with `NvAPI_VIO_SetConfig()` is shown in Code Listing 7. In the following Windows code example, a single dual-link stream is configured using a single channel from each of the first two jacks on the device with 4444 component sampling and 10 bits per component.

Code Listing 7: Configuring a Video Capture Device on Windows

```
// Now, set the config that we really want here.
memset(&l_vioConfig, 0, sizeof(l_vioConfig));
l_vioConfig.version = NVVIOCONFIG_VER;
l_vioConfig.nvvioConfigType = NVVIOCONFIGTYPE_IN;

// Signal Format
l_vioConfig.vioConfig.inConfig.signalFormat =
NVVIOSIGNALFORMAT_720P_60_00_SMPTE296;
l_vioConfig.fields = NVVIOCONFIG_SIGNALFORMAT;

// Streams - Define single 720p60 stream for now.
l_vioConfig.vioConfig.inConfig.numStreams = 1;
l_vioConfig.fields |= NVVIOCONFIG_STREAMS;
l_vioConfig.vioConfig.inConfig.numRawCaptureImages =
NVAPI_GVI_DEFAULT_RAW_CAPTURE_IMAGES;
l_vioConfig.vioConfig.inConfig.streams[0].sampling =
NVVIOCOMPONENTSAMPLING_4444;
l_vioConfig.vioConfig.inConfig.streams[0].bitsPerComponent = 10;
l_vioConfig.vioConfig.inConfig.streams[0].expansionEnable = 0;
l_vioConfig.vioConfig.inConfig.streams[0].numLinks = 2;
l_vioConfig.vioConfig.inConfig.streams[0].links[1].jack = 1;
l_vioConfig.vioConfig.inConfig.streams[0].links[1].channel = 0;
l_vioConfig.vioConfig.inConfig.streams[0].links[0].jack = 0;
l_vioConfig.vioConfig.inConfig.streams[0].links[0].channel = 0;

if ((NvAPI_VIO_SetConfig(m_vioHandle, &l_vioConfig) != NVAPI_OK)) {
      MessageBox(NULL, "Cannot set configuration of SDI input device.",
"Error", MB_OK);
      return E_FAIL;
}
```

On Linux, the stream configuration of the video device is set with `XNVCTRLStringOperation()` as demonstrated in Code Listing 8. Once the stream configuration is complete, the parameters of each stream are set with `XNVCTRLSetTargetAttribute()`. The following example configures a single dual-link stream with 4444 component sampling and 10 bits per component.

Code Listing 8: Configuring a Video Device on Linux

```
// Configure a sigle dual-link stream.
char instr[255];
char *outstr;
strcpy(instr, "stream=0, link0=jack0, link1=jack1");
if (!XNVCTRLStringOperation(dpy, NV_CTRL_TARGET_TYPE_GVI, 0, 0,
                       NV_CTRL_STRING_OPERATION_GVI_CONFIGURE_STREAMS,
                       instr, &outstr)) {
    fprintf(stderr, "Error configuring input ports as specified
streams\n");
}

XFree(outstr);

// Configure sampling for each stream.
for (int i = 0; i < op->numStreams; i++) {

    //
    // Set desired parameters
    //
    // Bits per component.
    XNVCTRLSetTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI, 0, i,

                NV_CTRL_GVI_REQUESTED_STREAM_BITS_PER_COMPONENT,
                        NV_CTRL_GVI_BITS_PER_COMPONENT_10);


    // Signal format.
    XNVCTRLSetTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI, 0, i,
                        NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT,
                        op->videoFormat);

    // Component sampling
    XNVCTRLSetTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI, 0, i,
            NV_CTRL_GVI_REQUESTED_STREAM_COMPONENT_SAMPLING,
                        NV_CTRL_GVI_COMPONENT_SAMPLING_4444);

    // Chroma sampling
    XNVCTRLSetTargetAttribute(dpy, NV_CTRL_TARGET_TYPE_GVI, 0, i,
                        NV_CTRL_GVI_REQUESTED_STREAM_CHROMA_EXPAND,
                        NV_CTRL_GVI_CHROMA_EXPAND_FALSE);
}
```

Note: Chroma expansion should be enabled when the captured format layout (capture device sampling) is 4:2:2 or 4:2:2:4 but the effective format layout (buffer or texture object sampling) will be 4:4:4 or 4:4:4:4.

Summary Note: A single dual-link stream compared to two single-link streams configuration table (Table 4-1).

A single dual-link stream compared to two single-link streams data layout table (Table 4-2).

Table 4-1. Single Dual-Link Stream Compared to Two Single-link Streams

	Dual-Link	Single-Link
Numstreams	1	2
Numlinks	2	1
Sampling	4:4:4, 4:4:4:4, 4:2:2:4	4:2:2

Table 4-2. Single Dual-Link Stream Compared to Two Single-link Streams Data Layout

	Dual-Link	Single-Link
Jack 0	YCRYCB	YCRYCB
Jack 1	CRCB or CRCBA or A	YCRYCB

5 DATA TRANSFER

Data transfer operations consist of capturing each of the incoming video streams into video buffer objects (VBO) or texture objects. The video data values captured can be transformed by a fixed-function color conversion before they are written into the destination buffer or texture object. This data transfer is achieved with the use of the `GL_NV_video_capture` OpenGL extension as described in this chapter. This extension performs the following tasks.

- Locking video capture device
- Binding video capture device
- Creation of VBOs or texture objects
- Binding VBOs or texture objects to the video device
- Initialization of data transfer parameters
- Start / Stop of data transfer
- Capture

Each of these steps will be described in detail. Unlike the device initialization and control described in Chapter 4, the OpenGL data transfer operations are very similar if not identical for both Windows and Linux thanks to the portability of OpenGL. As such, many of the code examples will apply to both operating systems.

5.1 LOCATING, LOCKING AND BINDING A VIDEO CAPTURE DEVICE

The first step in initializing the OpenGL data transfer is to locate, lock and then bind the video capture device which requires a current OpenGL rendering context. On Windows, an application must first enumerate all available video capture devices using `wglEnumerateVideoCaptureDevicesNV()`. If `numDevices` returned from this call is greater than 0, a subsequent call to the function returns a list of available video capture devices. An application must then loop through this list and find a device whose unique WGL ID matches the NVAPI VIO ID that we queried in Section 4.3. Once we have found a match, the application must call `wglLockVideoCaptureDeviceNV()` to lock the video capture device. A video capture device must be locked before it can be used. Once locked by a process, no other process can lock the same video capture device until the lock is released or the process ends. The following example simply chooses the first available video capture device. Once locked, the application must call `wglBindVideoCaptureDeviceNV()` to bind the locked video device to the desired video capture slot in the current OpenGL rendering context. This designated video capture slot must be greater or equal to 1.

Code Listing 9: Binding a Video Capture Device on Windows

```
// Enumerate the available video capture devices.
UINT numDevices = wglEnumerateVideoCaptureDevicesNV(m_hDC, NULL);
if (numDevices <= 0) {
    MessageBox(NULL, "wglEnumerateVideoDevicesNV() did not return any
devices.", "Error", MB_OK);
    return E_FAIL;
}

m_videoDevices = (HVIDEOINPUTDEVICENV*)malloc(numDevices *
                                           sizeof(m_videoDevices[0]));

if (!m_videoDevices) {
    fprintf(stderr, "malloc failed.  OOM?");
    return E_FAIL;
}

if (numDevices != wglEnumerateVideoCaptureDevicesNV(m_hDC,
                     m_videoDevices)) {
    free(m_videoDevices);
    MessageBox(NULL, "Inconsistent results from
wglEnumerateVideoDevicesNV()", "Error", MB_OK);
    return E_FAIL;
}

// Find an available device we can lock
for (UINT i=0; i< numDevices; ++i) {
    BOOL bLocked;
    bLocked = wglLockVideoCaptureDeviceNV(m_hDC, m_videoDevices[i]);
    if (bLocked) {
        m_device = m_videoDevices[i];
        break;
    }
}

free(m_videoDevices);

if (m_device == NULL) {
    // No lockable devices found
    MessageBox(NULL, "No lockable video capture device found.",
                  "Error", MB_OK);
        return E_FAIL;
}
//find a device whose WGL unique ID matches the NVAPI vioID
for (UINT i=0; i< numDevices; ++i) {
      int uniqueID;
      wglQueryVideoCaptureDeviceNV(hDc,m_videoDevices[i],
                                    WGL_UNIQUE_ID_NV, &uniqueID);
      if(uniqueID == vioID){
            m_device = m_videoDevices[i];
            break;
      }
}

free(m_videoDevices);

if(m_device == NULL)
{

      MessageBox(NULL, "No lockable video capture device found.",
                  "Error", MB_OK);
      return E_FAIL;
}
//Attempt to lock the found device
BOOL bLocked;
bLocked = wglLockVideoCaptureDeviceNV(m_hDC, m_device);if (!bLocked) {
      MessageBox(NULL, "No lockable video capture device found.",
                  "Error", MB_OK);

      return E_FAIL;
}

// wglBindVideoCaptureDeviceNV needs a context current
bRet = wglBindVideoCaptureDeviceNV(m_videoSlot, m_device);
assert(bRet && "Failed trying to bind the video capture device!");
```

On Linux, the procedure for enumerating, locking and then binding a video capture device is similar except that the corresponding GLX functions are utilized as shown. On Linux, once a video capture device is locked by a client, no other client can lock that video capture device until the lock is released, or the connection between the client holding the lock and the X server is broken.

Code Listing 10: Binding a Video Capture Device on Linux

```
// Enumerate available video capture devices
VideoInDevices = glXEnumerateVideoCaptureDevicesNV(dpy, screen,
                                                   &numDevices);

if (!VideoInDevices || numDevices <= 0) {
  printf("No video capture devices found.\n");
  return GL_FALSE;
}
//Find a device whose WGL unique ID matches the NVCtrl hGVI
for (UINT i=0; i< numDevices; ++i) {
      int uniqueID;
      glXQueryVideoCaptureDeviceNV(dpy, VideoDevices[i],
                                          GLX_UNIQUE_ID_NV, &uniqueID);
      if(uniqueID == hGVI){
            g_hVidInDevice = VideoDevices[i];
            break;
      }
}
XFree(ideoInDevices);
if(g_hVidInDevice == NULL)
{
  printf("No lockable video capture device found.\n");
  return GL_FALSE;
}

// Choose first device found.  Free device list.
g_hVidInDevice = VideoInDevices[0];
XFree(VideoInDevices);

// Lock video capture device.
glXLockVideoCaptureDeviceNV(dpy, g_hVidInDevice);

// Bind video capture device to the current OpenGL rendering context.
if (Success != glXBindVideoCaptureDeviceNV(dpy, gVideoSlot,
                                           g_hVidInDevice)) {
  printf("Could not bind video input device\n");
  return GL_FALSE;
}
```

5.2 PER-STREAM INITIALIZATION

Once the video capture device is bound, the next step is to setup the per-stream data transfer parameters. Use `glVideoCaptureStreamParameterfvNV()` to set these parameters. The following example shows the initialization of the color space conversion that will be performed by OpenGL in order to utilize YCrCb video data as an RGB texture. More information on color space conversion can be found in Chapter 6.

Code Listing 11: Specifying Color Space Conversion

```
// Setup CSC for each stream.
GLfloat mat[4][4];
float scale = 1.0f;

GLfloat max[] = {5000, 5000, 5000, 5000};;
GLfloat min[] = {0, 0, 0, 0};

// Initialize matrix to the identity.
mat[0][0] = scale; mat[0][1] = 0; mat[0][2] = 0; mat[0][3] = 0;
mat[1][0] = 0; mat[1][1] = scale; mat[1][2] = 0; mat[1][3] = 0;
mat[2][0] = 0; mat[2][1] = 0; mat[2][2] = scale; mat[2][3] = 0;
mat[3][0] = 0; mat[3][1] = 0; mat[3][2] = 0; mat[3][3] = scale;

GLfloat offset[] = {0, 0, 0, 0};

mat[0][0] = 1.164f *scale;
mat[0][1] = 1.164f *scale;
mat[0][2] = 1.164f *scale;
mat[0][3] = 0;

mat[1][0] = 0;
mat[1][1] = -0.392f *scale;
mat[1][2] = 2.017f *scale;
mat[1][3] = 0;

mat[2][0] = 1.596f *scale;
mat[2][1] = -0.813f *scale;
mat[2][2] = 0.f;
mat[2][3] = 0;

mat[3][0] = 0;
mat[3][1] = 0;
mat[3][2] = 0;
mat[3][3] = 1;

offset[0] =-0.87f;
offset[1] = 0.53026f;
offset[2] = -1.08f;
offset[3] = 0;
```

```
for  (int i=0; i < l_vioConfig.vioConfig.inConfig.numStreams; i++) {

glVideoCaptureStreamParameterfvNV(m_videoSlot, i,
                                   GL_VIDEO_COLOR_CONVERSION_MATRIX_NV,
                                   &mat[0][0]);
      assert(glGetError() == GL_NO_ERROR);

      glVideoCaptureStreamParameterfvNV(m_videoSlot, i,
                            GL_VIDEO_COLOR_CONVERSION_MAX_NV,&max[0]);
      assert(glGetError() == GL_NO_ERROR);

      glVideoCaptureStreamParameterfvNV(m_videoSlot, i,
                            GL_VIDEO_COLOR_CONVERSION_MIN_NV,&min[0]);
      assert(glGetError() == GL_NO_ERROR);

      glVideoCaptureStreamParameterfvNV(m_videoSlot, i,
                      GL_VIDEO_COLOR_CONVERSION_OFFSET_NV,&offset[0]);
      assert(glGetError() == GL_NO_ERROR);
}
```

5.3 BINDING VIDEO BUFFER OBJECTS

Captured video data is transferred into either video buffer objects or texture objects in GPU memory. Initialization of video buffer objects will be described in this section. See the next section for a description of texture object initialization.

Video buffer objects are created just like other OpenGL buffer objects by using `glGenBuffersARB()`. The internal format of the pixel data stored in the video buffer object is specified using `glVideoStreamParameterivNV()`. The possible internal data formats include:

- YCBYCR8_422_NV
- YCBAYCRA8_4224_NV
- Z6Y10Z6CB10Z6Y10Z6CR10_422_NV
- Z6Y10Z6CB10Z6A10Z6Y10Z6CR10Z6A10_4224_NV
- Z4Y12Z4CB12Z4Y12Z4CR12_422_NV
- Z4Y12Z4CB12Z4A12Z4Y12Z4CR12Z4A12_4224_NV
- Z4Y12Z4CB12Z4CR12_444_NV

A complete description of each of these formats can be found in Table 4.13 of the OpenGL extension specification.

Once the internal data format has been specified, space must be allocated for the captured video data prior to binding each buffer object to a video stream. This is done by first calling `glBindBufferARB()` to bind the buffer and then by calling glBufferDataARB() to specify the size as demonstrated Code Listing 12. The buffer pitch is determined by the internal data format specified for the buffer object and can be determined by calling glGetVideoCaptureStreamivNV() with the GL_VIDEO_BUFFER_PITCH_NV parameter. To bind a video buffer to a stream, call glBindVideoCaptureStreamBufferNV().

Code Listing 12: Video Buffer Specification

```
GLint videoBufferFormat = GL_RGBA8; // GL_YCBYCR8_422_NV;
GLint bufferPitch;

// Create video buffer objects
glGenBuffersARB(l_vioConfig.vioConfig.inConfig.numStreams,
                m_videoBuffer);
assert(glGetError() == GL_NO_ERROR);

// Allocate space in the buffer objects.
for (NvU32 i = 0; i < l_vioConfig.vioConfig.inConfig.numStreams; i++) {

      // Set the buffer object capture data format for each stream.
      glVideoCaptureStreamParameterivNV(m_videoSlot, i,
                                   GL_VIDEO_BUFFER_INTERNAL_FORMAT_NV,
                                   &videoBufferFormat);

      // Bind the buffer.
      glBindBufferARB(GL_VIDEO_BUFFER_NV, m_videoBuffer[i]);


      // Get the video buffer pitch
      glGetVideoCaptureStreamivNV(m_videoSlot, i,
                                  GL_VIDEO_BUFFER_PITCH_NV,
                                  &bufferPitch);

      // Allocate space in the buffer object
      glBufferDataARB(GL_VIDEO_BUFFER_NV, bufferPitch * videoHeight,
NULL,
                                    GL_STREAM_COPY_ARB);

      // Bind the buffer object to the video capture stream.
      glBindVideoCaptureStreamBufferNV(m_videoSlot, i, GL_FRAME_NV, 0);
}
```

Video Buffer Objects (VBO) can also be used to capture interlaced frames as individual fields. To do this, two calls to glBindVideoVideoCaptureStreamBufferNV have to be made. The following code example illustrates how to capture video in a stacked field mode.

Code Listing 12.1: Individual Field Capture

```
for (NvU32 i = 0; i < l_vioConfig.vioConfig.inConfig.numStreams; i++) {

    // Set the buffer object capture data format for each stream.
    glVideoCaptureStreamParameterivNV(m_videoSlot, i,
                                   GL_VIDEO_BUFFER_INTERNAL_FORMAT_NV,
                                   &videoBufferFormat);

    // Bind the buffer.
    glBindBufferARB(GL_VIDEO_BUFFER_NV, m_videoBuffer[i]);


    // Get the video buffer pitch
    glGetVideoCaptureStreamivNV(m_videoSlot, i,
                                GL_VIDEO_BUFFER_PITCH_NV,
                                &bufferPitch);

    // Allocate space in the buffer object
    glBufferDataARB(GL_VIDEO_BUFFER_NV, bufferPitch * m_videoHeight,
                    NULL,  GL_STREAM_READ_ARB);

    if(m_signalFormatDetail.signalFormat ==
       NVVIOSIGNALFORMAT_487I_59_94_SMPTE259_NTSC){
        glBindVideoCaptureStreamBufferNV(m_videoSlot, i,
                                         GL_FIELD_UPPER_NV, 0);
        glBindVideoCaptureStreamBufferNV(m_videoSlot, i,
                                         GL_FIELD_LOWER_NV,
                                 bufferPitch *((m_videoHeight>>1)+1));
    } else {

        glBindVideoCaptureStreamBufferNV(m_videoSlot, i,
                                         GL_FIELD_UPPER_NV, 0);
        glBindVideoCaptureStreamBufferNV(m_videoSlot, i,
                                         GL_FIELD_LOWER_NV,
                                      bufferPitch *(m_videoHeight>>1));
    }

}
```

5.4 BINDING VIDEO TEXTURES

Texture objects are created by calling `glGenTextures()`. Like in the case of buffer objects, a texture object is required for each stream to be captured. Texture objects are bound to each capture stream with `glBindVideoCaptureStreamTextureNV()`. Prior to binding a texture object, `glTexImage2D()` must be called to initialize the texture format and space required. Any other required texture parameters should be set with `glTexParameter()` as well. The following code example illustrates the binding of texture objects to the incoming video capture streams.

Code Listing 13: Video Texture Specification

```
// Create and initialize video texture objects.
glGenTextures(l_vioConfig.vioConfig.inConfig.numStreams,
m_videoTexture);


for (UINT i = 0; i < l_vioConfig.vioConfig.inConfig.numStreams; i++) {
glBindTexture(GL_TEXTURE_RECTANGLE_NV, m_videoTexture[i]);

      // Set texture parameters
      glTexParameterf(GL_TEXTURE_RECTANGLE_NV,
                     GL_TEXTURE_MIN_FILTER, GL_LINEAR);

      // Set texture format and size.
      glTexImage2D(GL_TEXTURE_RECTANGLE_NV, 0, GL_RGBA8, 1280, 720, 0,
                  GL_RGBA, GL_UNSIGNED_BYTE, NULL);

      // Bind the outputs for the stream
      glBindVideoCaptureStreamTextureNV(m_videoSlot, i, GL_FRAME_NV,
                                       GL_TEXTURE_RECTANGLE_NV,
                                        m_videoTexture[i]);

}
```

5.5 VIDEO CAPTURE

5.5.1 Starting Video Capture

Once the video capture streams have been specified and video buffer objects or video texture objects bound for each stream, the next step is to commence video capture. This is done by calling `glBeginVideoCaptureNV()`. At this point, the video capture device bound to the specified video slot will begin filling a queue of raw buffers with incoming video data.

Code Listing 14: Starting Video Capture

```
glBeginVideoCaptureNV(m_videoSlot);
```

5.5.2 Video Capture

Once video capture is started as described in Section 5.5.1, an application should call `glVideoCapture()` for each frame to be captured. At this time, pixel values for all streams on a device are transferred simultaneously to the buffer objects or textures bound for each stream. If no frames are available, this call will block until frames are ready for capture, or an error occurs.

The recommended way to structure an application for video capture is to set up a capture loop similar to a draw loop in a graphics rendering application or use a timer callback. This is illustrated in Code Listing 15.

Code Listing 15: Starting Video Capture

```
while(!done) {
    eval = glVideoCaptureNV(m_videoSlot, sequenceNum, timestamp);

    switch(eval) {
            case GL_SUCCESS_NV:
                  break;
            case GL_PARTIAL_SUCCESS_NV:
                  break;
            case GL_FAILURE_NV:
                  break;
            default:
                  break;
            }
} // while
```

The return value from `glVideoCapture()` will indicate the success or failure of the video capture and any errors that may have occurred. Following is a brief summary of the possible return values and errors that can be found.

GL_SUCCESS_NV

The capture operation completed successfully for all streams with objects bound.

GL_PARTIAL_SUCCESS_NV

The capture operation succeeded o some streams with objects bound.

GL_FAILURE_NV

The capture operation failed for all bound objects.

In the case that `glVideoCaptureNV()` returns `GL_SUCCESS_NV` or `GL_PARTIAL_SUCCESS_NV`, the sequenceNum parameters will be set to the sequence number of the frame captured starting at 0 for the first frame after glBeginVideoCaptureNV() was called. The parameter timestamp is set to the GPU time in nanoseconds that the video capture device began capturing the frame. The time at which glVideoCaptureNV() was called does not impact the value of timestamp returned, the time returned is the time at which the frame reached the video capture hardware. In the case the glVideoCaptureNV() returns GL_FAILURE_NV, the values of sequenceNum and timestamp are undefined.

5.5.2.1 Measuring Capture Latency

Capture latency of the application can be measured using the glGetInteger64v function with the GL_CURRENT_TIME_NV parameter.

Code Listing 15.1: Measuring Latency

```
GLuint64EXT captureTimeStart,captureTimeEnd;
ret = glVideoCaptureNV(m_videoSlot, sequenceNum, &captureTimeStart);
glGetInteger64v(GL_CURRENT_TIME_NV,(GLint64 *)&captureTimeEnd);
float captureLatency = (captureTimeEnd - captureTimeStart)*.000000001;
```

This will measure the time between when the frame arrived at the GVI device and when glVideoCaptureNV submitted the decode blit commands to the GPU. To include the time it took to do the decode blit, a glFinish call must be placed between the glVideoCaptureNV and glGetInteger64v calls. This will cause a stall in the OpenGL pipeline but will provide an accurate estimate of the time that it took for a captured video frame to become available on the GPU.

It is also possible to measure exclusively the amount of time it took to do a conversion blit on the GPU by using the OpenGL timer query object.

For video capture applications that do real or less than real time capture an undesired latency can appear because of buffering between the GPU and the capture device. Latency can be minimized by adjusting the size of the ring buffer (see Section 4.6). Executing a faster than real time capture will eliminate such latency. This can be achieved by repeatedly capturing frames until the measured latency is less than or equal to 1.5 times the frame time.

Code Listing 15.2: Eliminating Undesired Capture Latency

```
while(captureLatency > 1.5/frameRate)
{
  ret = glVideoCaptureNV(m_videoSlot, sequenceNum, &captureTimeStart);
  captureLatency = (captureTimeEnd - captureTimeStart)*.000000001;
};
```

5.5.3 Stopping Video Capture

Video capture is stopped by calling glEndVideoCaptureNV().

Code Listing 16: Stopping Video Capture

```
glEndVideoCaptureNV(m_videoSlot);
```

5.6 CLEANING UP

When video capture is complete and after `glEndVideoCaptureNV()` has been called, an application should unbind and release a video device. This is done by first unbinding the device by calling `wglBindVideoDeviceNV()` or `glxBindVideoDeviceNV()` with NULL as the device parameter. An application should then call `wglReleaseVideoCaptureDeviceNV()` or `glxReleaseVideoCaptureDeviceNV()` to release a video capture device.

Code Listing 16: Releasing a Video Device on Windows

```
HRESULT
cleanupVideo()
{
      BOOL bRet;

      // Unbind and release the capture device.
      bRet = wglBindVideoCaptureDeviceNV(1, NULL);
      assert(bRet && "Failed trying to unbind the video capture 
device!");

      // wglReleaseVideoCaptureDeviceNV should work even without a 
context current
      wglMakeCurrent(g_hDC, NULL);
       bRet = wglReleaseVideoCaptureDeviceNV(g_hDC, g_device);
      assert(bRet && "Failed trying to release the video capture 
device!");

      return S_OK;
}
```

Code Listing 17: Releasing a Video Device on Linux

```
GLvoid
cleanupVideo()
{
      // Pause/Stop capturing.
      glEndVideoCaptureNV(gVideoSlot);

      // Unbind the video capture device
      glxBindVideoCaptureDevice(dpy, gVideoSlot, NULL);

      // Release video capture device.
      glXReleaseVideoCaptureDeviceNV(dpy, g_hVidInDevice);

}
```

6 COLOR SPACE CONVERSION

6.1 OVERVIEW

Video data captured with a 4:4:4 or 4:4:4:4 sampling passes through a fixed-point color space conversion. Video pixels are transformed by the following equation as they are transferred from the Quadro SDI video capture device, to the bound video buffer or texture objects.

$$|Y_i|$$

$$C_{out} = clamp(\ M|Cr_i| + Offset)$$

$$|Cb_i|$$

$$|A_i|$$

In this equation, *M* and *Offset* represent the color space conversion matrix and color offset for the video capture stream. These are specified with `glVideoCaptureStreamParameterivNV()` with the parameters `GL_VIDEO_COLOR_CONVERSION_MATRIX_NV` and `GL_VIDEO_COLOR_CONVERSION_OFFSET_NV` as illustrated in Code Listing 11. `GL_VIDEO_COLOR_CONVERSION_MATRIX_NV` specifies a 16-value column-major order matrix. The *clamp* operation clamps each of the resulting components to the range [C_{min}, C_{max}]. C_{min} is the maximum of the corresponding component in the vector specified by `GL_VIDEO_COLOR_CONVERSION_MIN_NV` and the minimum value representable by the format of the destination surface. C_{max} is the minimum of the corresponding component in the vector specified by `GL_VIDEO_OLOR_CONVERSION_MAX_NV` and the maximum value representable by the destination format.

In the case that the destination surface utilizes a fixed-point or floating-point internal storage format, the captured video data will be converted to a floating point representation prior to color space conversion. The following equation describes this conversion.

$C_f = (C_i - D_{min}) / (D_{max} - D_{min})$

Where C_i is the color value of the incoming component. D_{min} and D_{max} are the minimum and maximum values of the video device and C_f is the resulting floating point color value.

Default values for color space conversion parameters are shown in Table 6-1.

Table 6-1. Default Values for Color Space Conversion Parameters

Parameter	Default Value
GL_VIDEO_COLOR_COVERSION_MATRIX_NV	Identity
GL_VIDEO_COLOR_CONVERSION_OFFSET_NV	[0 0 0 0]
GL_VIDEO_COLOR_CONVERSION_MIN_NV	[0 0 0 0]
GL_VIDEO_COLOR_CONVERSION_MAX_NV	[1 1 1 1]

6.2 TYPICAL COLOR SPACE CONVERSIONS

This section describes the matrix coefficients and scale and offset values for common video and full range color space conversions.

RGB [0,219] from ITU-R BT.601 Y'[0,219]CrCb[0,224]

=

RGB [0,255] from ITU-R BT.601 Y'[0,219]CrCb[0.224]

=

Offset (8-bit)
Y = 16/235 = 0.068
Cb = (240+16)/2 / (255) = 0.5
Cr = (240+16)/2 / (255) = 0.5

Scale (8-bit)
Y = (235-16) / 256 = 0.85546875
Cb = (240-16) / 256 = 0.875
Cr = (240-16) / 256 = 0.875

Offset (10-bit)
Y = 64/940 = 0.068
Cb = (960+64)/2 / (1023) = 0.5
Cr = (960+64)/2 / (1023) = 0.5

Scale (10-bit)
Y = (940-64) / 1024 = 0.85546875
Cb = (960-64) / 1024 = 0.875
Cr = (260-64) / 1024 = 0.875

RGB[0,255] from ITU-R BT.601 Y’CrCb[0,255]

=

RGB[0,255] from ITU-R BT.601 Y’CrCb[0,247]

=

Offset (8-bit)
Y = 4/251 = 0.015936
Cb = (251+4)/2 / (255) = 0.5
Cr = (251+4)/2 / (255) = 0.5

Offset(10-bit)
Y = 4/1019 = 0.0039254
Cb = (1019+4)/2 / (1023) = 0.5
Cr = (1019+4)/2 / (1023) = 0.5

Scale (8-bit)
Y = (251-4) / 256 = 0.964844
Cb = (251-4) / 256 = 0.964844
Cr = (251-4) / 256 = 0.964844

Scale (10-bit)
Y = (1019-4) / 1024 = 0.99121
Cb = (1019-4) / 1024 = 0.99121
Cr = (1019-4) / 1024 = 0.99121

RGB [0,219] from ITU-R BT.709 Y’[0,219]CrCb[0,224]

=

RGB [0,255] from ITU-R BT.709 Y’[0,219]CrCb[0,224]

=

Offset (8-bit)
Y = 16/235 = 0.068
Cb = (240+16)/2 / (255) = 0.5
Cr = (240+16)/2 / (255) = 0.5

Offset (10-bit)
Y = 64/940 = 0.068
Cb = (960+64)/2 / (1023) = 0.5
Cr = (960+64)/2 / (1023) = 0.5

Scale (8-bit)
Y = (235-16) / 256 = 0.85546875
Cb = (240-16) / 256 = 0.875
Cr = (240-16) / 256 = 0.875

Scale (10-bit)
Y = (940-64) / 1024 = 0.85546875
Cb = (960-64) / 1024 = 0.875
Cr = (260-64) / 1024 = 0.875

RGB[0,255] from ITU-R BT.709 [0,255]

=

RGB[0,255] from ITU-R BT.709 [0,247]

=

Offset (8-bit)
Y = 4/251 = 0.015936
Cb = (251+4)/2 / (255) = 0.5
Cr = (251+4)/2 / (255) = 0.5

Scale (8-bit)
Y = (251-4) / 256 = 0.964844
Cb = (251-4) / 256 = 0.964844
Cr = (251-4) / 256 = 0.964844

Offset(10-bit)
Y = 4/1019 = 0.0039254
Cb = (1019+4)/2 / (1023) = 0.5
Cr = (1019+4)/2 / (1023) = 0.5

Scale (10-bit)
Y = (1019-4) / 1024 = 0.99121
Cb = (1019-4) / 1024 = 0.99121
Cr = (1019-4) / 1024 = 0.99121

7 SDI OUTPUT

Captured video data from the Quadro SDI Capture device can be passed directly to an accompanying Quadro SDI output device through the use of the `GL_NV_present_video` OpenGL extension. This is done by first capturing the stream to a texture object and then passing the texture object to `glPresentVideoNV()` as illustrated in the previous code sample.

```
while(!done) {
eval = glVideoCaptureNV(m_videoSlot, sequenceNum, timestamp);

switch(eval) {
            case GL_SUCCESS_NV:
                  break;
            case GL_PARTIAL_SUCCESS_NV:
                  break;
            case GL_FAILURE_NV:
                  break;
            default:
                  break;
            }

// Send texture object to SDI device
glPresentFrameKeyedNV(1, 0,
                      presentTimeID, presentDurationID,
                      GL_FRAME_NV,
                      GL_TEXTURE_RECTANGLE_NV, gTO, 0,
                      GL_NONE, 0, 0);
} // while
```

For more information on programming the Quadro SDI Output device, see the *Quadro SDI Programming Guide.*

8 ANCILLARY DATA

The Quadro SDI Capture device captures the ancillary data in both the horizontal and vertical blanking regions of the video streams. Applications can access this data using the Ancillary Data API defined in `ANCapi` in the SDK. To utilize the API, applications designed for Microsoft Windows must statically link again `ANCapi.lib` while applications designed for Linux must link with `libanc.a`. The library files can be found in the appropriate lib folders in the SDK. The complete definition of the Ancillary Data API as defined in `ancapi.h` can be found in *Chapter 13 Ancillary Data API.*

8.1 GETTING STARTED

In order to capture ancillary data from the SDI device, the ancillary data API must be initialized. This initialization must be performed after the OpenGL initialization of the SDI device. Initialization is performed by calling the `initializeAncGVI()` function.

Code Listing 18: Initializing Ancillary Data API on Windows

```
// Initialize ANC API
if (NvVIOANCAPI_InitializeGVI(CNvSDIin::getHandle()) != NVAPI_OK) {
      return E_FAIL;
}
```

Code Listing 19: Initializing Ancillary Data API on Linux

```
// Initialize ANC API
if (NvVIOANCAPI_InitializeGVI(dpy, GVI_target_id) != NVAPI_OK){
      return false;
}
```

8.2 BASICS

Ancillary data is captured from the Quadro SDI device per frame into the corresponding fields in the structure below according to the setting of the `fields` mask that dictates those fields that should be captured.

```
// Per Frame
typedef struct tagNVVIOANCDATAFRAME {
    NvU32 version;                      // Structure version
    NvU32 fields;                      // Field mask
    NVVIOANCAUDIOGROUP  AudioGroup1; // Audio group 1
    NVVIOANCAUDIOGROUP  AudioGroup2; // Audio group 2
    NVVIOANCAUDIOGROUP  AudioGroup3; // Audio group 3
    NVVIOANCAUDIOGROUP  AudioGroup4; // Audio group 4
    NvU32 LTCTimecode;               // RP188
    NvU32 LTCUserBytes;
    NvU32 VITCTimecode;
    NvU32 VITCUserBytes;
    NvU32 FilmTimecode;
    NvU32 FilmUserBytes;
    NvU32 ProductionTimecode;        // RP201
    NvU32 ProductionUserBytes;       // RP201
    NvU32 FrameID;
    NvU32 numCustomPackets;
    NVVIOANCDATAPACKET *CustomPackets;
} NVVIOANCDATAFRAME;
```

Note: In the case of custom data packets and audio the application must allocate the required memory for captured data prior to passing the structure address to the API. Failure to properly allocate the required memory will result in a segmentation fault at runtime. It is important to properly set the fields bitmask to indicate only those ancillary data `fields` that an application is interested in. This prevents the ancillary data capture API from performing un-required work.

Once the fields mask has been set for the fields to be captured and any required memory allocated, an application should call the `NvVIOANCAPI_CaptureANCData()` function to capture the desired ancillary data into the proper fields. This is illustrated in the following code sample.

Note: As of Release 3.3 of the NVIDIA Quadro SDI SDK, only VITC, custom, and audio data packets are supported by the Ancillary Data API.

Code Listing 20: Capturing Ancillary Data

```
// Capture ANC data
NvVIOANCAPI_CaptureANCData(CNvSDIin::getHandle(), m_ancData);
```

This call should be made in the application capture loop after the call to `glVideoCaptureNV()`.

8.3 TIMECODE

In order to capture VITC timecode, the value of fields must be properly set to indicate to the ancillary data API that the timecode value should be returned.

Code Listing 21: Initialize Capture of VITC Timecode

```
for (int i = 0; i < NUM_STREAMS; i++) {
      m_ancData[i].fields |= NVVIOANCDATAFRAME_VITC;
}
```

The following code example illustrates the post-capture formatting of VITC timecode as defined by SMPTE 12M-1999. The time code data and relevant bit flags are captured into the 32-bit `VITCTimecode` field.

Code Listing 22: Parsing Captured VITC Timecode

```
// Draw timecode for first video stream
int frameTens = (m_ancData[0].VITCTimecode >> 4) & 0x3;
int frameUnits = m_ancData[0].VITCTimecode & 0xf;
int secondTens = (m_ancData[0].VITCTimecode >> 12) & 0x7;
int secondUnits = (m_ancData[0].VITCTimecode >> 8) & 0xf;
int minuteTens = (m_ancData[0].VITCTimecode >> 20) & 0x7;
int minuteUnits = (m_ancData[0].VITCTimecode >> 16) & 0xf;
int hourTens = (m_ancData[0].VITCTimecode >> 28)& 0x3;
int hourUnits = (m_ancData[0].VITCTimecode >> 24) & 0xf;

int len;
char buf[24];
sprintf(buf, "%d%d:%d%d:%d%d:%d%d", hourTens, hourUnits, minuteTens,
minuteUnits, secondTens, secondUnits, frameTens, frameUnits);
```

8.4 AUDIO

Audio data is captured as raw PCM audio samples formatted into data packets according to the SMPTE 272M for standard definition video signal formats or the SMPTE 299M specification for high definition video signal formats. The Quadro SDI capture device supports up to 16 channels of 24-bit audio at 48 kHz. The audio sample data as well as per-frame audio control data for each group of four channels is returned in an NVVIOANCAUDIOGROUP structure.

```
// Audio control
typedef struct tagNVVIOANCAUDIOCNTRL {
    NvU32 version;              // Structure version
    NvU8 frameNumber1_2;        // Frame number for channels 1 and 2
    NvU8 frameNumber3_4;        // Frame number for channels 3 and 4
    NVVIOANCAUDIO_SAMPLE_RATE rate;  // Audio sample rate
    NvU8 asynchronous;          // 0 = synchronous, 1 = asynchronous
    NvU8 activeChannels;   // Bitwise OR of active channel definitions
} NVVIOANCAUDIOCNTRL;

#define NVVIOANCAUDIOCNTRL_VERSION
MAKE_NVAPI_VERSION(NVVIOANCAUDIOCNTRL, 2)

// Audio group
typedef struct tagNVVIOANCAUDIOGROUP {
    NvU32 numAudioSamples; // Number of valid audio samples / channel
    NvU32  *audioData[4];     // Data pointer for audio channels 1-4
    NVVIOANCAUDIOCNTRL audioCntrl;   // Controls for audio channels 1-4
} NVVIOANCAUDIOGROUP;

#define NVVIOANCAUDIOGROUP_VERSION
MAKE_NVAPI_VERSION(NVVIOANCAUDIOGROUP, 1)
```

8.4.1 SMPTE 272M – Standard Definition Audio

Standard definition audio is captured as a 32-bit AES sub frame that contains up to 20 bits of audio data along with the block sync (Z), validity (V), user (U), channel (C), and parity (P) bits. Extracting the audio bits from the AES sub frame is demonstrated in Code Listing 23.

8.4.2 SMPTE 299M – High Definition Audio

High Definition audio samples are captured as a 32-bit value. This 32-bit value contains up to 24 bits audio data along with the block sync (Z), validity (V), user (U), channel (C), and parity (P) bits. Extracting the 24-bit audio sample data is illustrated in Code Listing 24.

8.4.3 Initialization

Prior to audio capture, the `fields` bitmask must be properly set to denote the desired channel groups to capture. An application must also allocate the required memory to hold the capture audio samples. This initialization is illustrated in Code Listing 23.

Code Listing 23: Initializing Audio Data Capture

```
// Determine the length of the audio sample sequence.
NvVIOANCAPI_NumAudioSamples(m_SDIin.getHandle(),
                              NVVIOANCAUDIO_SAMPLING_RATE_48_0,
                              (NvU32 *)&m_SequenceLength,
                              NULL);

// Allocate/reallocate required memory for the array to hold the number
// of  audio samples for each frame in a sequence.
if (m_NumAudioSamples)
      free(m_NumAudioSamples);
m_NumAudioSamples = (NvU32 *)calloc((size_t)m_SequenceLength,
sizeof(NvU32));

// Determine number of audio samples based on signal format
// and audio sampling rate.
NvVIOANCAPI_NumAudioSamples(m_SDIin.getHandle(),
                              NVVIOANCAUDIO_SAMPLING_RATE_48_0,
                              (NvU32 *)&m_SequenceLength,
                              (NvU32 *)m_NumAudioSamples);

// Determine the maximum number of audio sample for any given frame.
// Use this value when allocating space to store audio samples
NvU32 maxNumAudioSamples = 0;
for (unsigned int i = 0; i < m_SequenceLength; i++) {
      if (m_NumAudioSamples[i] > maxNumAudioSamples) {
            maxNumAudioSamples = m_NumAudioSamples[i];
      }
}

// Allocate space required for audio data packets for each
// audio group, four channels per group.  Space required depends
// upon signal format and audio rate.  Set bit field to indicate
// desired audio channels to capture.
for (int i = 0; i < m_numStreams; i++) {
      for (int j = 0; j < 4; j++) {
            m_ancData[i].AudioGroup1.numAudioSamples =
m_NumAudioSamples;
            m_ancData[i].AudioGroup1.audioData[j] = (NvU32
*)calloc(m_NumAudioSamples, sizeof(NvU32));
            m_ancData[i].fields |= NVVIOANCDATAFRAME_AUDIO_GROUP_1;

            m_ancData[i].AudioGroup2.numAudioSamples =
m_NumAudioSamples;
```

```
            m_ancData[i].AudioGroup2.audioData[j] = (NvU32
*)calloc(m_NumAudioSamples, sizeof(NvU32));
            m_ancData[i].fields |= NVVIOANCDATAFRAME_AUDIO_GROUP_2;

            m_ancData[i].AudioGroup3.numAudioSamples =
m_NumAudioSamples;
            m_ancData[i].AudioGroup3.audioData[j] = (NvU32
*)calloc(m_NumAudioSamples, sizeof(NvU32));
            m_ancData[i].fields |= NVVIOANCDATAFRAME_AUDIO_GROUP_3;

            m_ancData[i].AudioGroup4.numAudioSamples =
m_NumAudioSamples;
            m_ancData[i].AudioGroup4.audioData[j] = (NvU32
*)calloc(m_NumAudioSamples, sizeof(NvU32));
            m_ancData[i].fields |= NVVIOANCDATAFRAME_AUDIO_GROUP_4;
      }
}
```

The subsequent code sample shows how to extract the raw PCM audio data from the captured SMPTE 272 or 299 encoded audio samples.

Code Listing 24: Extracting Audio Data Samples

```
// Convert all SMPTE encoded audio samples in the current head
// buffer to 24-bit PCM audio samples.  In the case of SMPTE 272
// (NTSC and PAL) the 3 data UDWs containing 20-bit audio is packed
// into each 32-bit audio sample like this:
//
// Bits     Assignment
// ----     ----------
// 0  Z
// 1-2  ch 0 ch 1 - channel indicator bits
// 3-23 Audio information
// 24   V - validity bit
// 25   U - use bit
// 26   C - channel status
// 27   P - parity bit
//
//
// In the case of SMPTE 299 (HD), the 4 data UDWs containing
// upto 24-bit audio samples is packed into each 32-bit audio
// sample like this.
//
// Bits     Assignment
// ----     ----------
// 0-2  0
// 3    Z
// 4-27 Audio information
// 28   V - validity bit
// 39   U - use bit
// 30   C - channel status
// 31   P - parity bit
```

```
//

switch((NVVIOSIGNALFORMAT)m_configOptions.videoFormat) {
      case NVVIOSIGNALFORMAT_487I_59_94_SMPTE259_NTSC:
      case NVVIOSIGNALFORMAT_576I_50_00_SMPTE259_PAL:

            // SMPTE 272 case

            // For now, simply use 16 MSBs
*ptr++ = (sample1 >> 7) & 0xffff;
            *ptr++ = (sample2 >> 7) & 0xffff;

            break;

      default:

            // SMPTE 299 case

            // For now, simply use 16 MSBs
            *ptr++ = (sample1 >> 12) & 0xffff;
            *ptr++ = (sample2 >> 12) & 0xffff;

            break;

} // switch
```

8.5 CUSTOM DATA

Prior to capturing custom data packets an application must set the value of fields appropriately and allocate the required memory to hold the custom data.

Code Listing 25: Allocating Space for Custom Data Packets

```
// Allocate space for custom data.
for (int i = 0; i < NUM_STREAMS; i++) {
      m_ancData[i].fields |= NVVIOANCDATAFRAME_CUSTOM;
      m_ancData[i].numCustomPackets = 255;
      m_ancData[i].CustomPackets = (NVVIOANCDATAPACKET
*)calloc(m_ancData[i].numCustomPackets, sizeof(NVVIOANCDATAFRAME));
      }
```

Captured custom data complete with the custom data packet data identification (DID), secondary data identification (SDID) and the data count (DC) and checksum CS) are captured in the CustomPackets array in the `NVVIOANCDATAFRAME` structure.

8.6 CLEAN UP

Prior to application exit, the ancillary data API should be released. This is done by calling `NvVIOANCAPI_ReleaseGVI()`. The ancillary data API should also be released and reinitialized (after OpenGL re-initialization) when the video signal format changes. This is necessary in order for the state to be set properly for the new video signal format.

9 ADVANCED TOPICS

9.1 VIDEO CAPTURE IN A MULTI-GPU ENVIRONMENT

In video capture systems containing more than one GPU it might be beneficial to dedicate one of the GPUs for capture while others are reserved for video processing or other non-video tasks. Also, when dedicating a GPU for capture it's important to consider the PCI bandwidth requirements of the capture board. In current motherboard architectures 2 PCI slots can belong to 2 different north bridge chips which can be detrimental to GPU<->Capture board bandwidth.

On Windows, GPU affinity extension must be used to specify a device context corresponding to a particular GPU that the capture card can bind to.

Code Listing 26: Addressing a Particular GPU on Windows:

```
HGPUNV gpuList[MAX_GPUS];

//Populating a GPU affinity handle list.
int i = 0;
HGPUNV hGPU;
while(wglEnumGpusNV(GPUIdx,&hGPU))
{
      gpuList[i++] = hGPU;
}
int CaptureGPU = 0;
HGPUNV handles[2];
handles[0] = gpuList[CaptureGPU];
handles[1] = NULL;

HDC videoDC = wglCreateAffinityDCNV(handles);

//Using the affinity device context when setting up capture
UINT numDevices = wglEnumerateVideoCaptureDevicesNV(videoDC, NULL);
```

```
// See Code Listing #9 for the rest of the setup

//Create an OpenGL context associated with the above device context and
//make this context current for the OpenGL portion of the setup
```

On Linux, an XScreen associated with the chosen GPU must be used throughout capture configuration code. There might be cases where there is no one-to-one GPU<->XScreen correspondence in the system. NVCtrl API must be used to determine the GPU to XScreen mapping.

Code Listing 27: Addressing a Particular GPU on Linux

```
//Determine GPU<->XScreen mapping
ret = XNVCTRLQueryTargetCount(dpy, NV_CTRL_TARGET_TYPE_GPU, &num_gpus);
if (ret) {
    for (gpu = 0; gpu < num_gpus; gpu++) {
        /* X Screens driven by this GPU */
        ret = XNVCTRLQueryTargetBinaryData
            (dpy,
             NV_CTRL_TARGET_TYPE_GPU,
             gpu, // target_id
             0, // display_mask
             NV_CTRL_BINARY_DATA_XSCREENS_USING_GPU,
             (unsigned char **) &pData,
             &len);
        if (ret) {
            if(pData[0])
              xscreen[gpu] = pData[1];
        }
   }

//After an XScreen had been selected, it should be used in
// GL/GLX setup calls. Using the XScreen when setting up capture
VideoInDevices = glXEnumerateVideoCaptureDevicesNV(dpy, xscreen,
                                                   &numDevices);
// See Code Listing #10 for the rest of the setup

//Create an OpenGL context associated with the chosen XScreen and
//make this context current for the OpenGL portion of the setup
```

9.2 USING CUDA

Captured frames can be passed on to CUDA for processing. To do that, the CUDA device should be initialized for OpenGL interoperability. This can be done using `cuGLCtxCreate` call instead of the normal `cuCtxCreate`.

Note: An OpenGL capture context must be current before crating a CUDA context with OpenGL interoperability.

Code Listing 28: Creating CUDA Context for OpenGL Interoperability

```
CUdevice cuDevice;
CUcontext cuContext;
int selectedDevice = 0;
CUresult cerr = cuDeviceGet(&cuDevice, selectedDevice);
CheckError(cerr);
cerr = cuGLCtxCreate(&cuContext,
CU_CTX_MAP_HOST|CU_CTX_BLOCKING_SYNC,cuDevice);
CheckError(cerr);
```

A graphics object containing the video frame must be registered with CUDA in the beginning of program execution and mapped to CUDA address space every frame prior to CUDA's usage. The object must be unmapped before it can be used again for capture. The following example code illustrates this.

Code Listing 29: CUDA Processing of a Video Buffer Object Using CUDA Driver API

```
GLint buf = m_vidBufObj[objInd];

CUgraphicsResource  cudaResource;

//Registering is done only once in the beginning
cuGraphicsGLRegisterBuffer(&cudaResource, buf,
                           CU_GRAPHICS_MAP_RESOURCE_FLAGS_NONE);
unsigned char *dptr;
// Buffer object mapping:Done every frame
cuGraphicsMapResources(1, &cudaResource, 0);
size_t num_bytes;
cuGraphicsResourceGetMappedPointer((void**)&dptr, &num_bytes,
                                    cudaResource);

// Call the CUDA kernel here
// Buffer object unmapping:Done every frame
cuGraphicsUnmapResources(1, &cudaResource, 0);

// Unregistering is done only once in the end
cuGraphicsGLUnregisterBuffer(cudaResource)
```

CUDA – OpenGL interop does not require that the CUDA context and OpenGL context reside on the same device. When there are several GPUs present in the system, there is a possibility that the CUDA context and the OpenGL context reside on two separate devices. In this case the driver will move the buffer object from one device to the next via system memory every frame for CUDA OpenGL interop.

Unnecessary data movement can be avoided when the CUDA context and OpenGL context reside on the same GPU. On Windows it's possible to achieve this by using GPU affinity and `cuWGLGetDevice` as illustrated in the following code sample.

Code Listing 30: Using GPU Affinity for CUDA OpenGL Interoperability

```
HGPUNV gpuList[MAX_GPUS];
//See code listing 24 for gpuList setup
CUresult result = cuWGLGetDevice(&cuDevice, gpuList[CaptureGPU]);
CheckError(result);
// Now create the CUDA context
result = cuGLCtxCreate(&cuContext,
CU_CTX_MAP_HOST|CU_CTX_BLOCKING_SYNC,cuDevice);
CheckError(result);
```

Currently on Linux there is no counterpart to `cuWGLGetDevice` (such as `cuGLXGetDevice` call), but it is planned for a future release of CUDA. In leau of this, other techniques should be used to make sure that the CUDA context with OpenGL interop is created on the GPU with a particular XScreen. For example, one can make sure the capture GPU differs from the other GPUs in the system by name, then it would be possible to identify it in CUDA using `cuDeviceGetName` and only create a context with an OpenGL interop for a device that has a particular name.

9.3 MULTIPLE CAPTURE CARDS

The capture loop should always be gated by a single video timing. That's why it is important to consider whether all of the incoming signals are in sync or not when handling video that is being captured by multiple capture devices.

When all of the incoming video signals are in sync an application can have one capture loop and one OpenGL context that will capture all of the incoming video frames. This is possible by matching each capture device with a different video capture slot in the OpenGL context. For example, Card 1 can use Slot 1, Card 2 will use Slot 2 and so on.

In the case of the signals not being in sync, the application should allocate a separate capture thread with its own OpenGL context.

10 NV_VIDEO_CAPTURE

```
/* NV_video_capture */
#define GL_VIDEO_BUFFER_NV                                  0x9020
#define GL_VIDEO_BUFFER_BINDING_NV                          0x9021
#define GL_FIELD_UPPER_NV                                   0x9022
#define GL_FIELD_LOWER_NV                                   0x9023
#define GL_NUM_VIDEO_CAPTURE_STREAMS_NV                     0x9024
#define GL_NEXT_VIDEO_CAPTURE_BUFFER_STATUS_NV              0x9025
#define GL_VIDEO_CAPTURE_TO_422_SUPPORTED_NV                0x9026
#define GL_LAST_VIDEO_CAPTURE_STATUS_NV                     0x9027
#define GL_VIDEO_BUFFER_PITCH_NV                            0x9028
#define GL_VIDEO_COLOR_CONVERSION_MATRIX_NV                 0x9029
#define GL_VIDEO_COLOR_CONVERSION_MAX_NV                    0x902A
#define GL_VIDEO_COLOR_CONVERSION_MIN_NV                    0x902B
#define GL_VIDEO_COLOR_CONVERSION_OFFSET_NV                 0x902C
#define GL_VIDEO_BUFFER_INTERNAL_FORMAT_NV                  0x902D
#define GL_PARTIAL_SUCCESS_NV                               0x902E
#define GL_SUCCESS_NV                                       0x902F
#define GL_FAILURE_NV                                       0x9030
#define GL_YCBYCR8_422_NV                                   0x9031
#define GL_YCBAYCR8_4224_NV                                 0x9032
#define GL_Z6Y10Z6CB10Z6Y10Z6CR10_422_NV                    0x9033
#define GL_Z6Y10Z6CB10Z6A10Z6Y10Z6CR10Z6A10_4224_NV         0x9034
#define GL_Z4Y12Z4CB12Z4Y12Z4CR12_422_NV                    0x9035
#define GL_Z4Y12Z4CB12Z4A12Z4Y12Z4CR12Z4A12_4224_NV         0x9036
#define GL_Z4Y12Z4CB12Z4CR12_444_NV                         0x9037
#define GL_VIDEO_CAPTURE_FRAME_WIDTH_NV                     0x9038
#define GL_VIDEO_CAPTURE_FRAME_HEIGHT_NV                    0x9039
#define GL_VIDEO_CAPTURE_FIELD_UPPER_HEIGHT_NV              0x903A
#define GL_VIDEO_CAPTURE_FIELD_LOWER_HEIGHT_NV              0x903B


#ifndef GL_NV_video_capture
#define GL_NV_video_capture 1
typedef void (WINAPI * PFNGLBEGINVIDEOCAPTURENVPROC) (GLuint
video_capture_slot);
```

```
typedef void (WINAPI * PFNGLBINDVIDEOCAPTURESTREAMBUFFERNVPROC) (GLuint
video_capture_slot, GLuint stream, GLenum frame_region, GLintptr
offset);
typedef void (WINAPI * PFNGLBINDVIDEOCAPTURESTREAMTEXTURENVPROC)
(GLuint video_capture_slot, GLuint stream, GLenum frame_region, GLenum
target, GLuint texture);
typedef void (WINAPI * PFNGLENDVIDEOCAPTURENVPROC) (GLuint
video_capture_slot);
typedef void (WINAPI * PFNGLGETVIDEOCAPTUREIVNVPROC) (GLuint
video_capture_slot, GLenum pname, GLint *params);
typedef void (WINAPI * PFNGLGETVIDEOCAPTURESTREAMIVNVPROC) (GLuint
video_capture_slot, GLuint stream, GLenum pname, GLint *params);
typedef void (WINAPI * PFNGLGETVIDEOCAPTURESTREAMUIVNVPROC) (GLuint
video_capture_slot, GLuint stream, GLenum pname, GLuint *params);
typedef void (WINAPI * PFNGLGETVIDEOCAPTURESTREAMFVNVPROC) (GLuint
video_capture_slot, GLuint stream, GLenum pname, GLfloat *params);
typedef void (WINAPI * PFNGLGETVIDEOCAPTURESTREAMDVNVPROC) (GLuint
video_capture_slot, GLuint stream, GLenum pname, GLdouble *params);
typedef GLenum (WINAPI * PFNGLVIDEOCAPTURENVPROC) (GLuint
video_capture_slot, GLuint *sequence_num, GLuint64EXT *capture_time);
typedef void (WINAPI * PFNGLVIDEOCAPTURESTREAMPARAMETERIVNVPROC)
(GLuint video_capture_slot, GLuint stream, GLenum pname, const GLint
*params);
typedef void (WINAPI * PFNGLVIDEOCAPTURESTREAMPARAMETERUIVNVPROC)
(GLuint video_capture_slot, GLuint stream, GLenum pname, const GLuint
*params);
typedef void (WINAPI * PFNGLVIDEOCAPTURESTREAMPARAMETERFVNVPROC)
(GLuint video_capture_slot, GLuint stream, GLenum pname, const GLfloat
*params);
typedef void (WINAPI * PFNGLVIDEOCAPTURESTREAMPARAMETERDVNVPROC)
(GLuint video_capture_slot, GLuint stream, GLenum pname, const GLdouble
*params);
#endif

#ifndef WGL_NV_video_capture
#define WGL_NV_video_capture 1
typedef BOOL (WINAPI * PFNWGLBINDVIDEOCAPTUREDEVICENVPROC) (UINT
uVideoSlot, HVIDEOINPUTDEVICENV hVideoDevice);
typedef UINT (WINAPI * PFNWGLENUMERATEVIDEOCAPTUREDEVICESNVPROC) (HDC
hDC, HVIDEOINPUTDEVICENV *phDeviceList);
typedef BOOL (WINAPI * PFNWGLLOCKVIDEOCAPTUREDEVICENVPROC) (HDC hDC,
HVIDEOINPUTDEVICENV hDevice);
typedef BOOL (WINAPI * PFNWGLQUERYVIDEOCAPTUREDEVICENVPROC) (HDC hDC,
HVIDEOINPUTDEVICENV hDevice, INT iAttribute, INT *piValue);
typedef BOOL (WINAPI * PFNWGLRELEASEVIDEOCAPTUREDEVICENVPROC) (HDC hDC,
HVIDEOINPUTDEVICENV hDevice);
#endif

#ifndef GLX_NV_video_capture
#define GLX_NV_video_capture
extern GLXVideoCaptureDeviceNV*
    glXEnumerateVideoCaptureDevicesNV(Display *dpy,
                                      int screen,
                                      int *nelements);
```

```
extern void glXLockVideoCaptureDeviceNV(Display *dpy,
                                        GLXVideoCaptureDeviceNV
device);
extern void glXReleaseVideoCaptureDeviceNV(Display *dpy,
                                           GLXVideoCaptureDeviceNV
device);
extern int glXBindVideoCaptureDeviceNV(Display *dpy,
                                       unsigned int video_capture_slot,
                                       GLXVideoCaptureDeviceNV device);
extern int glXQueryVideoCaptureDeviceNV(Display *dpy,
                                        GLXVideoCaptureDeviceNV device,
                                        int attribute,
                                        int *value);
#endif
```

11 NVAPI VIO

```
typedef NvU32   NVVIOOWNERID;           // Unique identifier for VIO
owner (process identifier or NVVIOOWNERID_NONE)
#define NVVIOOWNERID_NONE                      0    // Unregistered
ownerId

typedef enum _NVVIOOWNERTYPE          // Owner type for device
{
    NVVIOOWNERTYPE_NONE,       //  No owner for device
    NVVIOOWNERTYPE_APPLICATIO,       //  Application owns device
    NVVIOOWNERTYPE_DESKTO,       //  Desktop transparent mode owns
device (not applicable for video input)
}NVVIOOWNERTYPE;

// Access rights for NvAPI_VIO_Open()
#define NVVIO_O_READ  0x00000000      // Read access             (not
applicable for video output)
#define NVVIO_O_WRITE_EXCLUSIVE 0x00010001      // Write exclusive
access  (not applicable for video input)

#define NVVIO_VALID_ACCESSRIGHTS            (NVVIO_O_READ
| \
                                             NVVIO_O_WRITE_EXCLUSIVE
)


// VIO_DATA.ulOwnerID high-bit is set only if device has been
initialized by VIOAPI
// examined at NvAPI_GetCapabilities|NvAPI_VIO_Open to determine if
settings need to be applied from registry or POR state read
#define NVVIO_OWNERID_INITIALIZED  0x80000000

// VIO_DATA.ulOwnerID next-bit is set only if device is currently in
exclusive write access mode from NvAPI_VIO_Open()
#define NVVIO_OWNERID_EXCLUSIVE    0x40000000

// VIO_DATA.ulOwnerID lower bits are:
```

```
//  NVGVOOWNERTYPE_xxx enumerations indicating use context
#define NVVIO_OWNERID_TYPEMASK     0x0FFFFFFF // mask for
NVVIOOWNERTYPE_xxx
//---------------------------------------------------------------------
// Enumerations
//---------------------------------------------------------------------

// Video signal format and resolution
typedef enum _NVVIOSIGNALFORMAT
{
    NVVIOSIGNALFORMAT_NONE,                                       //
Invalid signal format
    NVVIOSIGNALFORMAT_487I_59_94_SMPTE259_NTSC,     // 01  487i
59.94Hz  (SMPTE259) NTSC
    NVVIOSIGNALFORMAT_576I_50_00_SMPTE259_PAL,      // 02  576i
50.00Hz  (SMPTE259) PAL
    NVVIOSIGNALFORMAT_1035I_59_94_SMPTE260,         // 03  1035i
59.94Hz  (SMPTE260)
    NVVIOSIGNALFORMAT_1035I_60_00_SMPTE260,         // 04  1035i
60.00Hz  (SMPTE260)
    NVVIOSIGNALFORMAT_1080I_50_00_SMPTE295,         // 05  1080i
50.00Hz  (SMPTE295)
    NVVIOSIGNALFORMAT_1080I_60_00_SMPTE274,         // 06  1080i
60.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080I_59_94_SMPTE274,         // 07  1080i
59.94Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080I_50_00_SMPTE274,         // 08  1080i
50.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080P_30_00_SMPTE274,         // 09  1080p
30.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080P_29_97_SMPTE274,         // 10  1080p
29.97Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080P_25_00_SMPTE274,         // 11  1080p
25.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080P_24_00_SMPTE274,         // 12  1080p
24.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080P_23_976_SMPTE274,        // 13  1080p
23.976Hz (SMPTE274)
    NVVIOSIGNALFORMAT_720P_60_00_SMPTE296,          // 14  720p
60.00Hz  (SMPTE296)
    NVVIOSIGNALFORMAT_720P_59_94_SMPTE296,          // 15  720p
59.94Hz  (SMPTE296)
    NVVIOSIGNALFORMAT_720P_50_00_SMPTE296,          // 16  720p
50.00Hz  (SMPTE296)
    NVVIOSIGNALFORMAT_1080I_48_00_SMPTE274,         // 17  1080I
48.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080I_47_96_SMPTE274,         // 18  1080I
47.96Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_720P_30_00_SMPTE296,          // 19  720p
30.00Hz  (SMPTE296)
    NVVIOSIGNALFORMAT_720P_29_97_SMPTE296,          // 20  720p
29.97Hz  (SMPTE296)
    NVVIOSIGNALFORMAT_720P_25_00_SMPTE296,          // 21  720p
25.00Hz  (SMPTE296)
```

```
    NVVIOSIGNALFORMAT_720P_24_00_SMPTE296,          // 22  720p
24.00Hz  (SMPTE296)
    NVVIOSIGNALFORMAT_720P_23_98_SMPTE296,          // 23  720p
23.98Hz  (SMPTE296)
    NVVIOSIGNALFORMAT_2048P_30_00_SMPTE372,         // 24  2048p
30.00Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048P_29_97_SMPTE372,         // 25  2048p
29.97Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048I_60_00_SMPTE372,         // 26  2048i
60.00Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048I_59_94_SMPTE372,         // 27  2048i
59.94Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048P_25_00_SMPTE372,         // 28  2048p
25.00Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048I_50_00_SMPTE372,         // 29  2048i
50.00Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048P_24_00_SMPTE372,         // 30  2048p
24.00Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048P_23_98_SMPTE372,         // 31  2048p
23.98Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048I_48_00_SMPTE372,         // 32  2048i
48.00Hz  (SMPTE372)
    NVVIOSIGNALFORMAT_2048I_47_96_SMPTE372,         // 33  2048i
47.96Hz  (SMPTE372)

    NVVIOSIGNALFORMAT_1080PSF_25_00_SMPTE274,       // 34  1080PsF
25.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080PSF_29_97_SMPTE274,       // 35  1080PsF
29.97Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080PSF_30_00_SMPTE274,       // 36  1080PsF
30.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080PSF_24_00_SMPTE274,       // 37  1080PsF
24.00Hz  (SMPTE274)
    NVVIOSIGNALFORMAT_1080PSF_23_98_SMPTE274,       // 38  1080PsF
23.98Hz  (SMPTE274)

NVVIOSIGNALFORMAT_1080P_50_00_SMPTE274_3G_LEVEL_A, // 39  1080P
50.00Hz  (SMPTE274) 3G Level A
    NVVIOSIGNALFORMAT_1080P_59_94_SMPTE274_3G_LEVEL_A, // 40  1080P
59.94Hz  (SMPTE274) 3G Level A
    NVVIOSIGNALFORMAT_1080P_60_00_SMPTE274_3G_LEVEL_A, // 41  1080P
60.00Hz  (SMPTE274) 3G Level A

    NVVIOSIGNALFORMAT_1080P_60_00_SMPTE274_3G_LEVEL_B, // 42  1080p
60.00Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_1080I_60_00_SMPTE274_3G_LEVEL_B, // 43  1080i
60.00Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048I_60_00_SMPTE372_3G_LEVEL_B, // 44  2048i
60.00Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080P_50_00_SMPTE274_3G_LEVEL_B, // 45  1080p
50.00Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_1080I_50_00_SMPTE274_3G_LEVEL_B, // 46  1080i
50.00Hz  (SMPTE274) 3G Level B
```

```
    NVVIOSIGNALFORMAT_2048I_50_00_SMPTE372_3G_LEVEL_B, // 47  2048i
50.00Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080P_30_00_SMPTE274_3G_LEVEL_B, // 48  1080p
30.00Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048P_30_00_SMPTE372_3G_LEVEL_B, // 49  2048p
30.00Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080P_25_00_SMPTE274_3G_LEVEL_B, // 50  1080p
25.00Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048P_25_00_SMPTE372_3G_LEVEL_B, // 51  2048p
25.00Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080P_24_00_SMPTE274_3G_LEVEL_B, // 52  1080p
24.00Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048P_24_00_SMPTE372_3G_LEVEL_B, // 53  2048p
24.00Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080I_48_00_SMPTE274_3G_LEVEL_B, // 54  1080i
48.00Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048I_48_00_SMPTE372_3G_LEVEL_B, // 55  2048i
48.00Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080P_59_94_SMPTE274_3G_LEVEL_B, // 56  1080p
59.94Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_1080I_59_94_SMPTE274_3G_LEVEL_B, // 57  1080i
59.94Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048I_59_94_SMPTE372_3G_LEVEL_B, // 58  2048i
59.94Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080P_29_97_SMPTE274_3G_LEVEL_B, // 59  1080p
29.97Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048P_29_97_SMPTE372_3G_LEVEL_B, // 60  2048p
29.97Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080P_23_98_SMPTE274_3G_LEVEL_B, // 61  1080p
29.98Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048P_23_98_SMPTE372_3G_LEVEL_B, // 62  2048p
29.98Hz  (SMPTE372) 3G Level B
    NVVIOSIGNALFORMAT_1080I_47_96_SMPTE274_3G_LEVEL_B, // 63  1080i
47.96Hz  (SMPTE274) 3G Level B
    NVVIOSIGNALFORMAT_2048I_47_96_SMPTE372_3G_LEVEL_B, // 64  2048i
47.96Hz  (SMPTE372) 3G Level B


    NVVIOSIGNALFORMAT_END                           // 39  To indicate
end of signal format list
}NVVIOSIGNALFORMAT;

// SMPTE standards format
typedef enum _NVVIOVIDEOSTANDARD
{
    NVVIOVIDEOSTANDARD_SMPTE259,        // SMPTE259
    NVVIOVIDEOSTANDARD_SMPTE260,        // SMPTE260
    NVVIOVIDEOSTANDARD_SMPTE274,        // SMPTE274
    NVVIOVIDEOSTANDARD_SMPTE295,        // SMPTE295
    NVVIOVIDEOSTANDARD_SMPTE296,        // SMPTE296
    NVVIOVIDEOSTANDARD_SMPTE372,        // SMPTE372
}NVVIOVIDEOSTANDARD;

// HD or SD video type
```

```
typedef enum _NVVIOVIDEOTYPE
{
    NVVIOVIDEOTYPE_SD,       // Standard-definition (SD)
    NVVIOVIDEOTYPE_HD,       // High-definition     (HD)
}NVVIOVIDEOTYPE;

// Interlace mode
typedef enum _NVVIOINTERLACEMODE
{
    NVVIOINTERLACEMODE_PROGRESSIVE,       // Progressive
(p)
    NVVIOINTERLACEMODE_INTERLACE,       // Interlace
(i)
    NVVIOINTERLACEMODE_PSF,       // Progressive Segment Frame (psf)
}NVVIOINTERLACEMODE;

// Video data format
typedef enum _NVVIODATAFORMAT
{
    NVVIODATAFORMAT_UNKNOWN   = -1,       // Invalid DataFormat
    NVVIODATAFORMAT_R8G8B8_TO_YCRCB444,       // R8:G8:B8
=> YCrCb  (4:4:4)
    NVVIODATAFORMAT_R8G8B8A8_TO_YCRCBA4444,       // R8:G8:B8:A8
=> YCrCbA (4:4:4:4)
    NVVIODATAFORMAT_R8G8B8Z10_TO_YCRCBZ4444,       // R8:G8:B8:Z10
=> YCrCbZ (4:4:4:4)
    NVVIODATAFORMAT_R8G8B8_TO_YCRCB422,       // R8:G8:B8
=> YCrCb  (4:2:2)
    NVVIODATAFORMAT_R8G8B8A8_TO_YCRCBA4224,       // R8:G8:B8:A8
=> YCrCbA (4:2:2:4)
    NVVIODATAFORMAT_R8G8B8Z10_TO_YCRCBZ4224,       // R8:G8:B8:Z10
=> YCrCbZ (4:2:2:4)
    NVVIODATAFORMAT_X8X8X8_444_PASSTHRU,       // R8:G8:B8
=> RGB    (4:4:4)
    NVVIODATAFORMAT_X8X8X8A8_4444_PASSTHRU,       // R8:G8:B8:A8
=> RGBA   (4:4:4:4)
    NVVIODATAFORMAT_X8X8X8Z10_4444_PASSTHRU,       // R8:G8:B8:Z10
=> RGBZ   (4:4:4:4)
    NVVIODATAFORMAT_X10X10X10_444_PASSTHRU,       // Y10:CR10:CB10=>
YCrCb  (4:4:4)
    NVVIODATAFORMAT_X10X8X8_444_PASSTHRU,       // Y10:CR8:CB8=> YCrCb
(4:4:4)
    NVVIODATAFORMAT_X10X8X8A10_4444_PASSTHRU,       //
Y10:CR8:CB8:A10=> YCrCbA (4:4:4:4)
    NVVIODATAFORMAT_X10X8X8Z10_4444_PASSTHRU,       //
Y10:CR8:CB8:Z10=> YCrCbZ (4:4:4:4)
    NVVIODATAFORMAT_DUAL_R8G8B8_TO_DUAL_YCRCB422,       // R8:G8:B8 +
R8:G8:B8=> YCrCb  (4:2:2 + 4:2:2)
    NVVIODATAFORMAT_DUAL_X8X8X8_TO_DUAL_422_PASSTHRU,       //
Y8:CR8:CB8 + Y8:CR8:CB8 => YCrCb  (4:2:2 + 4:2:2)
    NVVIODATAFORMAT_R10G10B10_TO_YCRCB422,       // R10:G10:B10=> YCrCb
(4:2:2)
    NVVIODATAFORMAT_R10G10B10_TO_YCRCB444,       // R10:G10:B10=> YCrCb
(4:4:4)
```

```
    NVVIODATAFORMAT_Y12CR12CB12_TO_YCRCB444,        // Y12:CR12:CB12=>
YCrCb  (4:4:4)
    NVVIODATAFORMAT_Y12CR12CB12_TO_YCRCB422,        // Y12:CR12:CB12=>
YCrCb  (4:2:2)
    NVVIODATAFORMAT_Y10CR10CB10_TO_YCRCB422,        // Y10:CR10:CB10=>
YCrCb  (4:2:2)
    NVVIODATAFORMAT_Y8CR8CB8_TO_YCRCB422,        // Y8:CR8:CB8=> YCrCb
(4:2:2)
    NVVIODATAFORMAT_Y10CR8CB8A10_TO_YCRCBA4224,        //
Y10:CR8:CB8:A10=> YCrCbA (4:2:2:4)
    NVVIODATAFORMAT_R10G10B10_TO_RGB444,        // R10:G10:B10=> RGB
(4:4:4)
    NVVIODATAFORMAT_R12G12B12_TO_RGB444,        // R12:G12:B12=> RGB
(4:4:4)
}NVVIODATAFORMAT;

// Video output area
typedef enum _NVVIOOUTPUTAREA
{
    NVVIOOUTPUTAREA_FULLSIZE,        // Output to entire video
resolution (full size)
    NVVIOOUTPUTAREA_SAFEACTION,        // Output to centered 90% of
video resolution (safe action)
    NVVIOOUTPUTAREA_SAFETITLE,        // Output to centered 80% of video
resolution (safe title)
}NVVIOOUTPUTAREA;

// Synchronization source
typedef enum _NVVIOSYNCSOURCE
{
    NVVIOSYNCSOURCE_SDISYNC,        // SDI Sync  (Digital input)
    NVVIOSYNCSOURCE_COMPSYNC,        // COMP Sync (Composite input)
}NVVIOSYNCSOURCE;

// Composite synchronization type
typedef enum _NVVIOCOMPSYNCTYPE
{
    NVVIOCOMPSYNCTYPE_AUTO,        // Auto-detect
    NVVIOCOMPSYNCTYPE_BILEVEL,        // Bi-level signal
    NVVIOCOMPSYNCTYPE_TRILEVEL,        // Tri-level signal
}NVVIOCOMPSYNCTYPE;

// Video input output status
typedef enum _NVVIOINPUTOUTPUTSTATUS
{
    NVINPUTOUTPUTSTATUS_OFF,        // Not in use
    NVINPUTOUTPUTSTATUS_ERROR,        // Error detected
    NVINPUTOUTPUTSTATUS_SDI_SD,        // SDI (standard-definition)
    NVINPUTOUTPUTSTATUS_SDI_HD,        // SDI (high-definition)
}NVVIOINPUTOUTPUTSTATUS;

// Synchronization input status
typedef enum _NVVIOSYNCSTATUS
```

```
{
    NVVIOSYNCSTATUS_OFF,       // Sync not detected
    NVVIOSYNCSTATUS_ERROR,       // Error detected
    NVVIOSYNCSTATUS_SYNCLOSS,       // Genlock in use, format mismatch
with output
    NVVIOSYNCSTATUS_COMPOSITE,       // Composite sync
    NVVIOSYNCSTATUS_SDI_SD,       // SDI sync (standard-definition)
    NVVIOSYNCSTATUS_SDI_HD,       // SDI sync (high-definition)
}NVVIOSYNCSTATUS;

//Video Capture Status
typedef enum _NVVIOCAPTURESTATUS
{
    NVVIOSTATUS_STOPPED,       // Sync not detected
    NVVIOSTATUS_RUNNING,       // Error detected
    NVVIOSTATUS_ERROR,       // Genlock in use, format mismatch with
output
}NVVIOCAPTURESTATUS;

//Video Capture Status
typedef enum _NVVIOSTATUSTYPE
{
    NVVIOSTATUSTYPE_IN,       // Input Status
    NVVIOSTATUSTYPE_OUT,       // Output Status
}NVVIOSTATUSTYPE;

#define NVAPI_MAX_VIO_DEVICES                 8   // Assumption,
maximum 4 SDI input and 4 SDI output cards supported on a system
#define NVAPI_MAX_VIO_JACKS                   4   // 4 physical jacks
supported on each SDI input card.
#define NVAPI_MAX_VIO_CHANNELS_PER_JACK       2   // Each physical jack
an on SDI input card can have
                                                  // two "channels" in
the case of "3G" VideoFormats, as specified
                                                  // by SMPTE 425; for
non-3G VideoFormats, only the first channel within
                                                  // a physical jack is
valid
#define NVAPI_MAX_VIO_STREAMS                 4   // 4 Streams, 1 per
physical jack
#define NVAPI_MIN_VIO_STREAMS                 1
#define NVAPI_MAX_VIO_LINKS_PER_STREAM        2   // SDI input supports
a max of 2 links per stream
#define NVAPI_MAX_FRAMELOCK_MAPPING_MODES     20
#define NVAPI_GVI_MIN_RAW_CAPTURE_IMAGES      1   // Min number of
capture images
#define NVAPI_GVI_MAX_RAW_CAPTURE_IMAGES      32  // Max number of
capture images
#define NVAPI_GVI_DEFAULT_RAW_CAPTURE_IMAGES  5   // Default number of
capture images

// Data Signal notification events. These need a event handler in RM.
// Register/Unregister and PopEvent NVAPI's are already available.
// Device configuration
typedef enum _NVVIOCONFIGTYPE
{
    NVVIOCONFIGTYPE_IN,       // Input Status
    NVVIOCONFIGTYPE_OUT,       // Output Status
}NVVIOCONFIGTYPE;

typedef enum _NVVIOCOLORSPACE
{
    NVVIOCOLORSPACE_UNKNOWN,
    NVVIOCOLORSPACE_YCBCR,
    NVVIOCOLORSPACE_YCBCRA,
    NVVIOCOLORSPACE_YCBCRD,
    NVVIOCOLORSPACE_GBR,
    NVVIOCOLORSPACE_GBRA,
    NVVIOCOLORSPACE_GBRD,
} NVVIOCOLORSPACE;

// Component sampling
typedef enum _NVVIOCOMPONENTSAMPLING
{
    NVVIOCOMPONENTSAMPLING_UNKNOWN,
    NVVIOCOMPONENTSAMPLING_4444,
    NVVIOCOMPONENTSAMPLING_4224,
    NVVIOCOMPONENTSAMPLING_444,
    NVVIOCOMPONENTSAMPLING_422
} NVVIOCOMPONENTSAMPLING;

typedef enum _NVVIOBITSPERCOMPONENT
{
    NVVIOBITSPERCOMPONENT_UNKNOWN,
    NVVIOBITSPERCOMPONENT_8,
    NVVIOBITSPERCOMPONENT_10,
    NVVIOBITSPERCOMPONENT_12,
} NVVIOBITSPERCOMPONENT;

typedef enum _NVVIOLINKID
{
    NVVIOLINKID_UNKNOWN,
    NVVIOLINKID_A,
    NVVIOLINKID_B,
    NVVIOLINKID_C,
    NVVIOLINKID_D
} NVVIOLINKID;

//---------------------------------------------------------------------
// Structures
//---------------------------------------------------------------------

#define NVVIOCAPS_VIDOUT_SDI                0x00000001      // Supports
Serial Digital Interface (SDI) output
```

```
#define NVVIOCAPS_SYNC_INTERNAL             0x00000100      // Supports
Internal timing source
#define NVVIOCAPS_SYNC_GENLOCK              0x00000200      // Supports
Genlock timing source
#define NVVIOCAPS_SYNCSRC_SDI               0x00001000      // Supports
Serial Digital Interface (SDI) synchronization input
#define NVVIOCAPS_SYNCSRC_COMP              0x00002000      // Supports
Composite synchronization input
#define NVVIOCAPS_OUTPUTMODE_DESKTOP        0x00010000      // Supports
Desktop transparent mode
#define NVVIOCAPS_OUTPUTMODE_OPENGL         0x00020000      // Supports
OpenGL application mode
#define NVVIOCAPS_VIDIN_SDI                 0x00100000      // Supports
Serial Digital Interface (SDI) input

#define NVVIOCLASS_SDI                      0x00000001      // SDI-
class interface: SDI output with two genlock inputs

// Device capabilities
typedef struct _NVVIOCAPS
{
    NvU32             version;                              //
Structure version
    NvAPI_String      adapterName;                          // Graphics
adapter name
    NvU32             adapterClass;                         // Graphics
adapter classes (NVVIOCLASS_SDI mask)
    NvU32             adapterCaps;                          // Graphics
adapter capabilities (NVVIOCAPS_* mask)
    NvU32             dipSwitch;                            // On-board
DIP switch settings bits
    NvU32             dipSwitchReserved;                    // On-board
DIP switch settings reserved bits
    NvU32             boardID;                              // Board ID
    struct                                                  //
    {                                                       // Driver
version
        NvU32          majorVersion;                        // Major
version
        NvU32          minorVersion;                        // Minor
version
    } driver;                                               //
    struct                                                  //
    {                                                       // Firmware
version
        NvU32          majorVersion;                        // Major
version
        NvU32          minorVersion;                        // Minor
version
    } firmWare;                                             //
    NVVIOOWNERID      ownerId;                              // Unique
identifier for owner of video output (NVVIOOWNERID_INVALID if free
running)
```

```
    NVVIOOWNERTYPE    ownerType;                             // Owner
type (OpenGL application or Desktop mode)
} NVVIOCAPS;

#define NVVIOCAPS_VER   MAKE_NVAPI_VERSION(NVVIOCAPS,1)

// Input channel status
typedef struct _NVVIOCHANNELSTATUS
{
    NvU32                    smpte352;                        // 4-byte
SMPTE 352 video payload identifier
    NVVIOSIGNALFORMAT        signalFormat;                    // Signal
format
    NVVIOBITSPERCOMPONENT    bitsPerComponent;                // Bits
per component
    NVVIOCOMPONENTSAMPLING samplingFormat;                    //
Sampling format
    NVVIOCOLORSPACE          colorSpace;                      // Color
space
    NVVIOLINKID              linkID;                          // Link ID
} NVVIOCHANNELSTATUS;

// Input device status
typedef struct _NVVIOINPUTSTATUS
{
    NVVIOCHANNELSTATUS
vidIn[NVAPI_MAX_VIO_JACKS][NVAPI_MAX_VIO_CHANNELS_PER_JACK];     //
Video input status per channel within a jack
    NVVIOCAPTURESTATUS       captureStatus;                 // status of
video capture
} NVVIOINPUTSTATUS;

// Output device status
typedef struct _NVVIOOUTPUTSTATUS
{
    NVVIOINPUTOUTPUTSTATUS vid1Out;                         // Video 1
output status
    NVVIOINPUTOUTPUTSTATUS vid2Out;                         // Video 2
output status
    NVVIOSYNCSTATUS sdiSyncIn;                       // SDI sync input
status
    NVVIOSYNCSTATUS compSyncIn;                      // Composite sync
input status
    NvU32 syncEnable;                      // Sync enable (TRUE if using
syncSource)
    NVVIOSYNCSOURCE syncSource;                      // Sync source
    NVVIOSIGNALFORMAT syncFormat;                      // Sync format
    NvU32 frameLockEnable;                 // Framelock enable flag
    NvU32 outputVideoLocked;               // Output locked status
    NvU32 dataIntegrityCheckErrorCount;    // Data integrity check error
count
    NvU32 dataIntegrityCheckEnabled;       // Data integrity check
status enabled
```

```
    NvU32 dataIntegrityCheckFailed;       // Data integrity check
status failed
    NvU32 uSyncSourceLocked;              // genlocked to framelocked
to ref signal
    NvU32 uPowerOn;                       // TRUE: indicates there is
sufficient power
} NVVIOOUTPUTSTATUS;

// Video device status.
typedef struct _NVVIOSTATUS
{
    NvU32 version;                        // Structure version
    NVVIOSTATUSTYPE nvvioStatusType;                // Input or Output
status
    union
    {
        NVVIOINPUTSTATUS  inStatus;  //  Input device status
        NVVIOOUTPUTSTATUS outStatus; //  Output device status
    }vioStatus;
} NVVIOSTATUS;

#define NVVIOSTATUS_VER   MAKE_NVAPI_VERSION(NVVIOSTATUS,1)

// Output region
typedef struct _NVVIOOUTPUTREGION
{
    NvU32 x;                // Horizontal origin in pixels
    NvU32 y;                // Vertical origin in pixels
    NvU32 width;            // Width of region in pixels
    NvU32 height;           // Height of region in pixels
} NVVIOOUTPUTREGION;

// Gamma ramp (8-bit index)
typedef struct _NVVIOGAMMARAMP8
{
    NvU16 uRed[256];   // Red channel gamma ramp (8-bit index, 16-bit
values)
    NvU16 uGreen[256]; // Green channel gamma ramp (8-bit index, 16-bit
values)
    NvU16 uBlue[256];  // Blue channel gamma ramp (8-bit index, 16-bit
values)
} NVVIOGAMMARAMP8;

// Gamma ramp (10-bit index)
typedef struct _NVVIOGAMMARAMP10
{
    NvU16 uRed[1024];  // Red channel gamma ramp (10-bit index, 16-bit
values)
    NvU16 uGreen[1024]; // Green channel gamma ramp (10-bit index, 16-
bit values)
    NvU16 uBlue[1024];  // Blue channel gamma ramp (10-bit index, 16-
bit values)
} NVVIOGAMMARAMP10;
```

```
// Sync delay
typedef struct _NVVIOSYNCDELAY
{
    NvU32 version;      // Structure version
    NvU32 horizontalDelay; // Horizontal delay in pixels
    NvU32 verticalDelay;  // Vertical delay in lines
} NVVIOSYNCDELAY;

#define NVVIOSYNCDELAY_VER   MAKE_NVAPI_VERSION(NVVIOSYNCDELAY,1)


// Video mode information
typedef struct _NVVIOVIDEOMODE
{
    NvU32 horizontalPixels;                   // Horizontal resolution
(in pixels)
    NvU32 verticalLines;                      // Vertical resolution
for frame (in lines)
    float fFrameRate;                         // Frame rate
    NVVIOINTERLACEMODE   interlaceMode;                      //
Interlace mode
    NVVIOVIDEOSTANDARD   videoStandard;                      // SMPTE
standards format
    NVVIOVIDEOTYPE       videoType;                          // HD or
SD signal classification
} NVVIOVIDEOMODE;

// Signal format details
typedef struct _NVVIOSIGNALFORMATDETAIL
{
    NVVIOSIGNALFORMAT    signalFormat;                       // Signal
format enumerated value
    NVVIOVIDEOMODE       videoMode;                          // Video
mode for signal format
}NVVIOSIGNALFORMATDETAIL;

// Buffer formats
#define NVVIOBUFFERFORMAT_R8G8B8                  0x00000001   //
R8:G8:B8
#define NVVIOBUFFERFORMAT_R8G8B8Z24               0x00000002   //
R8:G8:B8:Z24
#define NVVIOBUFFERFORMAT_R8G8B8A8                0x00000004   //
R8:G8:B8:A8
#define NVVIOBUFFERFORMAT_R8G8B8A8Z24             0x00000008   //
R8:G8:B8:A8:Z24
#define NVVIOBUFFERFORMAT_R16FPG16FPB16FP         0x00000010   //
R16FP:G16FP:B16FP
#define NVVIOBUFFERFORMAT_R16FPG16FPB16FPZ24      0x00000020   //
R16FP:G16FP:B16FP:Z24
#define NVVIOBUFFERFORMAT_R16FPG16FPB16FPA16FP    0x00000040   //
R16FP:G16FP:B16FP:A16FP
```

```
#define NVVIOBUFFERFORMAT_R16FPG16FPB16FPA16FPZ24 0x00000080   //
R16FP:G16FP:B16FP:A16FP:Z24

// Data format details
typedef struct _NVVIODATAFORMATDETAIL
{
    NVVIODATAFORMAT   dataFormat;                              // Data
format enumerated value
    NvU32             vioCaps;                                 // Data
format capabilities (NVVIOCAPS_* mask)
}NVVIODATAFORMATDETAIL;

// Colorspace conversion
typedef struct _NVVIOCOLORCONVERSION
{
    NvU32       version;                                    //
Structure version
    float       colorMatrix[3][3];                          //
Output[n] =
    float       colorOffset[3];                             //
Input[0] * colorMatrix[n][0] +
    float       colorScale[3];                              //
Input[1] * colorMatrix[n][1] +
                                                            //
Input[2] * colorMatrix[n][2] +
                                                            //
OutputRange * colorOffset[n]
                                                            //  where
OutputRange is the standard magnitude of
                                                            //
Output[n][n] and colorMatrix and colorOffset
                                                            //  values
are within the range -1.0 to +1.0
    NvU32      compositeSafe;                               //
compositeSafe constrains luminance range when using composite output
} NVVIOCOLORCONVERSION;

#define NVVIOCOLORCONVERSION_VER
MAKE_NVAPI_VERSION(NVVIOCOLORCONVERSION,1)

// Gamma correction
typedef struct _NVVIOGAMMACORRECTION
{
    NvU32            version;                               //
Structure version
    NvU32            vioGammaCorrectionType;                // Gamma
correction type (8-bit or 10-bit)
    union                                                   // Gamma
correction:
    {
        NVVIOGAMMARAMP8  gammaRamp8;                        // Gamma
ramp (8-bit index, 16-bit values)
        NVVIOGAMMARAMP10 gammaRamp10;                       // Gamma
ramp (10-bit index, 16-bit values)
```

```
    }gammaRamp;
    float            fGammaValueR;
      // Red Gamma value within gamma ranges. 0.5 - 6.0
    float            fGammaValueG;
      // Green Gamma value within gamma ranges. 0.5 - 6.0
    float            fGammaValueB;
      // Blue Gamma value within gamma ranges. 0.5 - 6.0
} NVVIOGAMMACORRECTION;

#define NVVIOGAMMACORRECTION_VER
MAKE_NVAPI_VERSION(NVVIOGAMMACORRECTION,1)

#define MAX_NUM_COMPOSITE_RANGE      2                      // maximum
number of ranges per channel

typedef struct _NVVIOCOMPOSITERANGE
{
    NvU32   uRange;
    NvU32   uEnabled;
    NvU32   uMin;
    NvU32   uMax;
} NVVIOCOMPOSITERANGE;


// Device configuration (fields masks indicating NVVIOCONFIG fields to
use for NvVioGet/Set/Test/CreateDefaultConfig())
#define NVVIOCONFIG_SIGNALFORMAT            0x00000001      //
dwFields: signalFormat
#define NVVIOCONFIG_DATAFORMAT              0x00000002      //
dwFields: dataFormat
#define NVVIOCONFIG_OUTPUTREGION            0x00000004      //
dwFields: outputRegion
#define NVVIOCONFIG_OUTPUTAREA              0x00000008      //
dwFields: outputArea
#define NVVIOCONFIG_COLORCONVERSION         0x00000010      //
dwFields: colorConversion
#define NVVIOCONFIG_GAMMACORRECTION         0x00000020      //
dwFields: gammaCorrection
#define NVVIOCONFIG_SYNCSOURCEENABLE        0x00000040      //
dwFields: syncSource and syncEnable
#define NVVIOCONFIG_SYNCDELAY               0x00000080      //
dwFields: syncDelay
#define NVVIOCONFIG_COMPOSITESYNCTYPE       0x00000100      //
dwFields: compositeSyncType
#define NVVIOCONFIG_FRAMELOCKENABLE         0x00000200      //
dwFields: EnableFramelock
#define NVVIOCONFIG_422FILTER               0x00000400      //
dwFields: bEnable422Filter
#define NVVIOCONFIG_COMPOSITETERMINATE      0x00000800      //
dwFields: bCompositeTerminate
#define NVVIOCONFIG_DATAINTEGRITYCHECK      0x00001000      //
dwFields: bEnableDataIntegrityCheck
#define NVVIOCONFIG_CSCOVERRIDE             0x00002000      //
dwFields: colorConversion override
```

```
#define NVVIOCONFIG_FLIPQUEUELENGTH         0x00004000      //
dwFields: flipqueuelength control
#define NVVIOCONFIG_ANCTIMECODEGENERATION   0x00008000      //
dwFields: bEnableANCTimeCodeGeneration
#define NVVIOCONFIG_COMPOSITE               0x00010000      //
dwFields: bEnableComposite
#define NVVIOCONFIG_ALPHAKEYCOMPOSITE       0x00020000      //
dwFields: bEnableAlphaKeyComposite
#define NVVIOCONFIG_COMPOSITE_Y             0x00040000      //
dwFields: compRange
#define NVVIOCONFIG_COMPOSITE_CR            0x00080000      //
dwFields: compRange
#define NVVIOCONFIG_COMPOSITE_CB            0x00100000      //
dwFields: compRange
#define NVVIOCONFIG_FULL_COLOR_RANGE        0x00200000      //
dwFields: bEnableFullColorRange
#define NVVIOCONFIG_RGB_DATA                0x00400000      //
dwFields: bEnableRGBData
#define NVVIOCONFIG_RESERVED_SDIOUTPUTENABLE         0x00800000      //
dwFields: bEnableSDIOutput
#define NVVIOCONFIG_STREAMS                 0x01000000      //
dwFields: streams

// Don't forget to update NVVIOCONFIG_VALIDFIELDS in
NvVIOApiInternals.h when NVVIOCONFIG_ALLFIELDS changes.
#define NVVIOCONFIG_ALLFIELDS   ( NVVIOCONFIG_SIGNALFORMAT          | \
                                  NVVIOCONFIG_DATAFORMAT            | \
                                  NVVIOCONFIG_OUTPUTREGION          | \
                                  NVVIOCONFIG_OUTPUTAREA            | \
                                  NVVIOCONFIG_COLORCONVERSION       | \
                                  NVVIOCONFIG_GAMMACORRECTION       | \
                                  NVVIOCONFIG_SYNCSOURCEENABLE      | \
                                  NVVIOCONFIG_SYNCDELAY             | \
                                  NVVIOCONFIG_COMPOSITESYNCTYPE     | \
                                  NVVIOCONFIG_FRAMELOCKENABLE       | \
                                  NVVIOCONFIG_422FILTER             | \
                                  NVVIOCONFIG_COMPOSITETERMINATE    | \
                                  NVVIOCONFIG_DATAINTEGRITYCHECK    | \
                                  NVVIOCONFIG_CSCOVERRIDE           | \
                                  NVVIOCONFIG_FLIPQUEUELENGTH       | \
                                  NVVIOCONFIG_ANCTIMECODEGENERATION | \
                                  NVVIOCONFIG_COMPOSITE             | \
                                  NVVIOCONFIG_ALPHAKEYCOMPOSITE     | \
                                  NVVIOCONFIG_COMPOSITE_Y           | \
                                  NVVIOCONFIG_COMPOSITE_CR          | \
                                  NVVIOCONFIG_COMPOSITE_CB          | \
                                  NVVIOCONFIG_FULL_COLOR_RANGE      | \
                                  NVVIOCONFIG_RGB_DATA              | \
                                  NVVIOCONFIG_RESERVED_SDIOUTPUTENABLE
| \
                                  NVVIOCONFIG_STREAMS)
```

```
#define NVVIOCONFIG_VALIDFIELDS  ( NVVIOCONFIG_SIGNALFORMAT          |
\
                                   NVVIOCONFIG_DATAFORMAT            |
\
                                   NVVIOCONFIG_OUTPUTREGION          |
\
                                   NVVIOCONFIG_OUTPUTAREA            |
\
                                   NVVIOCONFIG_COLORCONVERSION       |
\
                                   NVVIOCONFIG_GAMMACORRECTION       |
\
                                   NVVIOCONFIG_SYNCSOURCEENABLE      |
\
                                   NVVIOCONFIG_SYNCDELAY             |
\
                                   NVVIOCONFIG_COMPOSITESYNCTYPE     |
\
                                   NVVIOCONFIG_FRAMELOCKENABLE       |
\
                                   NVVIOCONFIG_RESERVED_SDIOUTPUTENABLE
| \
                                   NVVIOCONFIG_422FILTER             |
\
                                   NVVIOCONFIG_COMPOSITETERMINATE    |
\
                                   NVVIOCONFIG_DATAINTEGRITYCHECK    |
\
                                   NVVIOCONFIG_CSCOVERRIDE           |
\
                                   NVVIOCONFIG_FLIPQUEUELENGTH       |
\
                                   NVVIOCONFIG_ANCTIMECODEGENERATION |
\
                                   NVVIOCONFIG_COMPOSITE             |
\
                                   NVVIOCONFIG_ALPHAKEYCOMPOSITE     |
\
                                   NVVIOCONFIG_COMPOSITE_Y           |
\
                                   NVVIOCONFIG_COMPOSITE_CR          |
\
                                   NVVIOCONFIG_COMPOSITE_CB          |
\
                                   NVVIOCONFIG_FULL_COLOR_RANGE      |
\
                                   NVVIOCONFIG_RGB_DATA              |
\
                                   NVVIOCONFIG_RESERVED_SDIOUTPUTENABLE
| \
                                   NVVIOCONFIG_STREAMS)

#define NVVIOCONFIG_DRIVERFIELDS ( NVVIOCONFIG_OUTPUTREGION          |
\
                                   NVVIOCONFIG_OUTPUTAREA             |
\
                                   NVVIOCONFIG_COLORCONVERSION        |
\
                                   NVVIOCONFIG_FLIPQUEUELENGTH)

#define NVVIOCONFIG_GAMMAFIELDS  ( NVVIOCONFIG_GAMMACORRECTION        )

#define NVVIOCONFIG_RMCTRLFIELDS ( NVVIOCONFIG_SIGNALFORMAT           |
\
                                   NVVIOCONFIG_DATAFORMAT             |
\
                                   NVVIOCONFIG_SYNCSOURCEENABLE       |
\
                                   NVVIOCONFIG_COMPOSITESYNCTYPE      |
\
                                   NVVIOCONFIG_FRAMELOCKENABLE        |
\
                                   NVVIOCONFIG_422FILTER              |
\
                                   NVVIOCONFIG_COMPOSITETERMINATE     |
\
                                   NVVIOCONFIG_DATAINTEGRITYCHECK     |
\
                                   NVVIOCONFIG_COMPOSITE              |
\
                                   NVVIOCONFIG_ALPHAKEYCOMPOSITE      |
\
                                   NVVIOCONFIG_COMPOSITE_Y            |
\
                                   NVVIOCONFIG_COMPOSITE_CR           |
\
                                   NVVIOCONFIG_COMPOSITE_CB)

#define NVVIOCONFIG_RMSKEWFIELDS ( NVVIOCONFIG_SYNCDELAY              )

#define NVVIOCONFIG_ALLOWSDIRUNNING_FIELDS (
NVVIOCONFIG_DATAINTEGRITYCHECK     | \
                                             NVVIOCONFIG_SYNCDELAY
| \
                                             NVVIOCONFIG_CSCOVERRIDE
| \

NVVIOCONFIG_ANCTIMECODEGENERATION  | \
                                             NVVIOCONFIG_COMPOSITE
| \

NVVIOCONFIG_ALPHAKEYCOMPOSITE      | \
                                             NVVIOCONFIG_COMPOSITE_Y
| \
                                             NVVIOCONFIG_COMPOSITE_CR
| \
                                             NVVIOCONFIG_COMPOSITE_CB)
```

```
 #define NVVIOCONFIG_RMMODESET_FIELDS ( NVVIOCONFIG_SIGNALFORMAT
| \
                                        NVVIOCONFIG_DATAFORMAT
| \
                                        NVVIOCONFIG_SYNCSOURCEENABLE
| \
                                        NVVIOCONFIG_FRAMELOCKENABLE
| \
                                        NVVIOCONFIG_COMPOSITESYNCTYPE )


// Output device configuration
// No members can be deleted from below structure. Only add new members
at the
// end of the structure
typedef struct _NVVIOOUTPUTCONFIG
{
    NVVIOSIGNALFORMAT    signalFormat;                         //
Signal format for video output
    NVVIODATAFORMAT      dataFormat;                           // Data
format for video output
    NVVIOOUTPUTREGION    outputRegion;                         //
Region for video output (Desktop mode)
    NVVIOOUTPUTAREA      outputArea;                           //
Usable resolution for video output (safe area)
    NVVIOCOLORCONVERSION colorConversion;                      // Color
conversion.
    NVVIOGAMMACORRECTION gammaCorrection;
    NvU32                syncEnable;                           // Sync
enable (TRUE to use syncSource)
    NVVIOSYNCSOURCE      syncSource;                           // Sync
source
    NVVIOSYNCDELAY       syncDelay;                            // Sync
delay
    NVVIOCOMPSYNCTYPE    compositeSyncType;                    //
Composite sync type
    NvU32                frameLockEnable;                      // Flag
indicating whether framelock was on/off
    NvU32                psfSignalFormat;                      //
Inidcates whether contained format is PSF Signal format
    NvU32                enable422Filter;                      //
Enables/Disables 4:2:2 filter
    NvU32                compositeTerminate;                   //
Composite termination
    NvU32                enableDataIntegrityCheck;             //
Enable data integrity check: true - enable, false - disable
    NvU32                cscOverride;                          // Use
provided CSC color matrix to overwrite
    NvU32                flipQueueLength;                      //
Number of buffers used for the internal flipqueue
    NvU32                enableANCTimeCodeGeneration;          //
Enable SDI ANC time code generation
    NvU32                enableComposite;                      //
Enable composite
```

```
    NvU32                enableAlphaKeyComposite;              // 
Enable Alpha key composite
    NVVIOCOMPOSITERANGE  compRange;                            // 
Composite ranges
    NvU8                 reservedData[256];                    // 
Inicates last stored SDI output state TRUE-ON / FALSE-OFF
    NvU32                enableFullColorRange;                 // Flag 
indicating Full Color Range
    NvU32                enableRGBData;                        // 
Indicates data is in RGB format
} NVVIOOUTPUTCONFIG;

// Stream configuration
typedef struct _NVVIOSTREAM
{
    NvU32 bitsPerComponent;                     // Bits per component
    NVVIOCOMPONENTSAMPLING  sampling;    // Sampling
    NvU32 expansionEnable;                      // Enable/disable 
4:2:2->4:4:4 expansion
    NvU32 numLinks;  // Number of active links
    struct
    {
        NvU32 jack;                                 // This stream's 
link[i] will use the specified (0-based) channel within the
        NvU32 channel;                              // specified (0-
based) jack
    } links[NVAPI_MAX_VIO_LINKS_PER_STREAM];
} NVVIOSTREAM;

// Input device configuration
typedef struct _NVVIOINPUTCONFIG
{
    NvU32                numRawCaptureImages;                  // 
numRawCaptureImages is the number of frames to keep in the capture 
queue.
                                                               // must 
be between NVAPI_GVI_MIN_RAW_CAPTURE_IMAGES and 
NVAPI_GVI_MAX_RAW_CAPTURE_IMAGES,
    NVVIOSIGNALFORMAT    signalFormat;                         // 
Signal format.
                                                               // 
Please note that both numRawCaptureImages and signalFormat should be 
set together.
    NvU32                numStreams;                           // 
Number of active streams.
    NVVIOSTREAM          streams[NVAPI_MAX_VIO_STREAMS];       // 
Stream configurations
} NVVIOINPUTCONFIG;

typedef struct _NVVIOCONFIG
{
    NvU32 version;                              // Structure version
    NvU32 fields;                               // Caller sets to 
NVVIOCONFIG_* mask for fields to use
```

```
    NVVIOCONFIGTYPE  nvvioConfigType;                      // Input or
Output configuration
    union
    {
        NVVIOINPUTCONFIG  inConfig;                            //
Input device configuration
        NVVIOOUTPUTCONFIG outConfig;                           //
Output device configuration
    }vioConfig;
} NVVIOCONFIG;

#define NVVIOCONFIG_VER   MAKE_NVAPI_VERSION(NVVIOCONFIG,1)

typedef struct
{
    NvPhysicalGpuHandle                         hPhysicalGpu;
              //handle to Physical GPU (This could be NULL for GVI
device if its not binded)
    NvVioHandle                         hVioHandle;
//handle to SDI Input/Output device
    NvU32                               vioId;
//device Id of SDI Input/Output device
    NvU32                               outputId;
            //deviceMask of the SDI display connected to GVO device.

//outputId will be 0 for GVI device.
} NVVIOTOPOLOGYTARGET;


typedef struct _NV_VIO_TOPOLOGY
{
    NvU32                       version;
    NvU32                       vioTotalDeviceCount;
//How many vio targets are valid
    NVVIOTOPOLOGYTARGET         vioTarget[NVAPI_MAX_VIO_DEVICES];
//Array of vio targets
}NV_VIO_TOPOLOGY, NVVIOTOPOLOGY;

#define NV_VIO_TOPOLOGY_VER  MAKE_NVAPI_VERSION(NV_VIO_TOPOLOGY,1)
#define NVVIOTOPOLOGY_VER    MAKE_NVAPI_VERSION(NVVIOTOPOLOGY,1)

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_GetCapabilities
//
// Description: Determine graphics adapter video I/O capabilities.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN] - The caller provides the SDI device
handle as input.
//              pAdapterCaps[OUT] - Pointer to receive capabilities
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - NVVIOCAPS struct
version used by the app is not compatible
//              NVAPI_NOT_SUPPORTED                - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_GetCapabilities(NvVioHandle hVioHandle,
                                          NVVIOCAPS *pAdapterCaps);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_Open
//
// Description: Open graphics adapter for video I/O operations
//              using the OpenGL application interface.  Read
operations
//              are permitted in this mode by multiple clients, but
Write
//              operations are application exclusive.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]     - The caller provides the SDI
output device handle as input.
//              vioClass[IN]        - Class interface (NVVIOCLASS_*
value)
//              ownerType[IN]       - user should specify the ownerType
( NVVIOOWNERTYPE_APPLICATION or NVVIOOWNERTYPE_DESKTOP)
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_NOT_SUPPORTED                - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_DEVICE_BUSY                 - Access denied for
requested access
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_Open(NvVioHandle       hVioHandle,
                               NvU32             vioClass,
                               NVVIOOWNERTYPE    ownerType);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_Close
//
// Description: Closes graphics adapter for Graphics to Video
operations
//              using the OpenGL application interface.  Closing an
//              OpenGL handle releases the device.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]  - The caller provides the SDI output
device handle as input.
//              bRelease         - boolean value to decide on keeping
or releasing ownership
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_NOT_SUPPORTED                - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_DEVICE_BUSY                 - Access denied for
requested access
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_Close(NvVioHandle       hVioHandle,
                                NvU32             bRelease);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_Status
//
// Description: Get Video I/O LED status.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN] - The caller provides the SDI device
handle as input.
//              pStatus(OUT)   - returns pointer to the NVVIOSTATUS
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Invalid structure
version
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_Status(NvVioHandle     hVioHandle,
                                 NVVIOSTATUS     *pStatus);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_SyncFormatDetect
//
// Description: Detects Video I/O incoming sync video format.
//
```

```
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN] - The caller provides the SDI device
handle as input.
//              pWait(OUT)     - Pointer to receive milliseconds to
wait
//                               before VIOStatus will return detected
//                               syncFormat.
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_SyncFormatDetect(NvVioHandle hVioHandle,
                                           NvU32       *pWait);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_GetConfig
//
// Description: Get Graphics to Video configuration.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN] - The caller provides the SDI device
handle as input.
//              pConfig(OUT)    - Pointer to Graphics to Video
configuration
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Invalid structure
version
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_GetConfig(NvVioHandle        hVioHandle,
                                    NVVIOCONFIG        *pConfig);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_SetConfig
//
// Description: Set Graphics to Video configuration.
//
//  SUPPORTED OS: Windows XP and higher
```

```
//
// Parameters:  NvVioHandle[IN]     - The caller provides the SDI
device handle as input.
//              pConfig(IN)         - Pointer to Graphics to Video
config
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Stucture version
invalid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_DEVICE_BUSY                 - Access denied for
requested access
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_SetConfig(NvVioHandle            hVioHandle,
                                    const NVVIOCONFIG      *pConfig);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_SetCSC
//
// Description: Set colorspace conversion parameters.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]     - The caller provides the SDI
device handle as input.
//              pCSC(IN)            - Pointer to CSC parameters
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Stucture version
invalid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_DEVICE_BUSY                 - Access denied for
requested access
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_SetCSC(NvVioHandle           hVioHandle,
                                 NVVIOCOLORCONVERSION  *pCSC);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_GetCSC
//
// Description: Get colorspace conversion parameters.
```

```
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]     - The caller provides the SDI
device handle as input.
//              pCSC(OUT)           - Pointer to CSC parameters
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Stucture version
invalid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_GetCSC(NvVioHandle           hVioHandle,
                                 NVVIOCOLORCONVERSION  *pCSC);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_SetGamma
//
// Description: Set gamma conversion parameters.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]     - The caller provides the SDI
device handle as input.
//              pGamma(IN)          - Pointer to gamma parameters
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Stucture version
invalid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_DEVICE_BUSY                 - Access denied for
requested access
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_SetGamma(NvVioHandle           hVioHandle,
                                   NVVIOGAMMACORRECTION  *pGamma);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_GetGamma
//
// Description: Get gamma conversion parameters.
//
```

```
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]     - The caller provides the SDI
device handle as input.
//              pGamma(OUT)         - Pointer to gamma parameters
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Stucture version
invalid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//-----------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_GetGamma(NvVioHandle           hVioHandle,
                                   NVVIOGAMMACORRECTION* pGamma);

//-----------------------------------------------------------------------
// Function:    NvAPI_VIO_SetSyncDelay
//
// Description: Set sync delay parameters.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN] - The caller provides the SDI device
handle as input.
//              pSyncDelay(IN)  - const Pointer to sync delay
parameters
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Stucture version
invalid
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_DEVICE_BUSY                 - Access denied for
requested access
//-----------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_SetSyncDelay(NvVioHandle hVioHandle,
                                   const NVVIOSYNCDELAY   *pSyncDelay);

//-----------------------------------------------------------------------
// Function:    NvAPI_VIO_GetSyncDelay
//
// Description: Get sync delay parameters.
//
//  SUPPORTED OS: Windows XP and higher
//
```

```
// Parameters:  NvVioHandle[IN]     - The caller provides the SDI
device handle as input.
//              pSyncDelay(OUT)     - Pointer to sync delay parameters
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Stucture version
invalid
//              NVAPI_ERROR                       - NVAPI Random errors
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_GetSyncDelay(NvVioHandle hVioHandle,
                                       NVVIOSYNCDELAY   *pSyncDelay);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_IsRunning
//
// Description: Determine if Video I/O is running.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]            - The caller provides the
SDI device handle as input.
//
// Returns:     NVAPI_DRIVER_RUNNING       - Video I/O running
//              NVAPI_DRIVER_NOTRUNNING    - Video I/O not running
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_IsRunning(NvVioHandle hVioHandle);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_Start
//
// Description: Start Video I/O.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]    - The caller provides the SDI device
handle as input.
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_DEVICE_BUSY                 - Access denied for
requested access
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_Start(NvVioHandle hVioHandle);
```

```
//---------------------------------------------------------------------
// Function:    NvAPI_VIO_Stop
//
// Description: Stop Video I/O.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]    - The caller provides the SDI device
handle as input.
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_DEVICE_BUSY                 - Access denied for
requested access
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_Stop(NvVioHandle hVioHandle);


//---------------------------------------------------------------------
// Function:    NvAPI_VIO_IsFrameLockModeCompatible
//
// Description: Checkes whether modes are compatible in framelock mode
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]         - The caller provides the SDI
device handle as input.
//              srcEnumIndex(IN)        - Source Enumeration index
//              destEnumIndex(IN)       - Destination Enumeration index
//              pbCompatible(OUT)       - Pointer to receive
compatability
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_NOT_SUPPORTED               - Video I/O not
supported
//              NVAPI_ERROR                       - NVAPI Random errors
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_IsFrameLockModeCompatible(NvVioHandle
                                                    hVioHandle,
                                                    NvU32
                                                    srcEnumIndex,
                                                    NvU32
                                                    destEnumIndex,
                                                    NvU32*
                                                    pbCompatible);

//-----------------------------------------------------------------------
// Function:    NvAPI_VIO_EnumDevices
//
// Description: Enumerate all valid SDI topologies
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[OUT]                  - User passes the
pointer of NvVioHandle[] array to get handles to all the connected vio
devices.
//              vioDeviceCount[OUT]               - User gets total
number of VIO devices connected to the system.
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_ERROR                       - NVAPI Random errors
//              NVAPI_NVIDIA_DEVICE_NOT_FOUND     - No SDI Device found
//-----------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_EnumDevices(NvVioHandle
                                     hVioHandle[NVAPI_MAX_VIO_DEVICES],
                                      Nv32 *vioDeviceCount);



//-----------------------------------------------------------------------
// Function:    NvAPI_VIO_QueryTopology
//
// Description: Enumerate all valid SDI topologies
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  pNvVIOTopology[OUT]       - User passes the pointer to
NVVIOTOPOLOGY to fetch all valid SDI Topologies.
//
// Returns:     NVAPI_OK                          - Success
//              NVAPI_API_NOT_INTIALIZED          - NVAPI Not
Initialized
//              NVAPI_INVALID_ARGUMENT            - Arguments passed to
API are not valid
//              NVAPI_INCOMPATIBLE_STRUCT_VERSION - Invalid structure
version
//              NVAPI_ERROR                       - NVAPI Random errors
//-----------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_QueryTopology(NV_VIO_TOPOLOGY
                                        *pNvVIOTopology);
```

```
//---------------------------------------------------------------------
// Function:    NvAPI_VIO_EnumSignalFormats
//
// Description: Enumerate signal formats supported by Video I/O.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]          - The caller provides the SDI
device handle as input.
//              enumIndex(IN)            - Enumeration index
//              pSignalFormatDetail(OUT) - Pointer to receive detail or
NULL
//
// Returns:     NVAPI_OK                 - Success
//              NVAPI_END_ENUMERATION    - No more signal formats to
enumerate
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_EnumSignalFormats(NvVioHandle hVioHandle,
                                            NvU32 enumIndex,
                                            NVVIOSIGNALFORMATDETAIL
                                            *pSignalFormatDetail);

//---------------------------------------------------------------------
// Function:    NvAPI_VIO_EnumDataFormats
//
// Description: Enumerate data formats supported by Video I/O.
//
//  SUPPORTED OS: Windows XP and higher
//
// Parameters:  NvVioHandle[IN]        - The caller provides the SDI
device handle as input.
//              enumIndex(IN)          - Enumeration index
//              pDataFormatDetail(OUT) - Pointer to receive detail or
NULL
//
// Returns:     NVAPI_OK               - Success
//              NVAPI_END_ENUMERATION  - No more data formats to
enumerate
//              NVAPI_NOT_SUPPORTED    - Unsupported NVVIODATAFORMAT_
enumeration
//---------------------------------------------------------------------
NVAPI_INTERFACE NvAPI_VIO_EnumDataFormats(NvVioHandle hVioHandle,
                                          NvU32 enumIndex,
                                          NVVIODATAFORMATDETAIL
                                          *pDataFormatDetail);
```

12 NV CONTROL VIO CONTROLS

```
/**********************************************************************
****/

/*
 * Attribute Targets
 *
 * Targets define attribute groups.  For example, some attributes are
only
 * valid to set on a GPU, others are only valid when talking about an
 * X Screen.  Target types are then what is used to identify the target
 * group of the attribute you wish to set/query.
 *
 * Here are the supported target types:
 */

#define NV_CTRL_TARGET_TYPE_X_SCREEN   0
#define NV_CTRL_TARGET_TYPE_GPU        1
#define NV_CTRL_TARGET_TYPE_FRAMELOCK  2
#define NV_CTRL_TARGET_TYPE_VCSC       3 /* Visual Computing System */
#define NV_CTRL_TARGET_TYPE_GVI        4


/**********************************************************************
****/

/*
 * Attributes
 *
 * Some attributes may only be read; some may require a display_mask
 * argument and others may be valid only for specific target types.
 * This information is encoded in the "permission" comment after each
 * attribute #define, and can be queried at run time with
 * XNVCTRLQueryValidAttributeValues() and/or
 * XNVCTRLQueryValidTargetAttributeValues()
 *
 * Key to Integer Attribute "Permissions":
 *
 * R: The attribute is readable (in general, all attributes will be
 *    readable)
 *
 * W: The attribute is writable (attributes may not be writable for
 *    various reasons: they represent static system information, they
 *    can only be changed by changing an XF86Config option, etc).
 *
 * D: The attribute requires the display mask argument.  The
 *    attributes NV_CTRL_CONNECTED_DISPLAYS and
NV_CTRL_ENABLED_DISPLAYS
 *    will be a bitmask of what display devices are connected and what
 *    display devices are enabled for use in X, respectively.  Each bit
 *    in the bitmask represents a display device; it is these bits
which
 *    should be used as the display_mask when dealing with attributes
 *    designated with "D" below.  For attributes that do not require
the
 *    display mask, the argument is ignored.
 *
 * G: The attribute may be queried using an NV_CTRL_TARGET_TYPE_GPU
 *    target type via XNVCTRLQueryTargetAttribute().
 *
 * F: The attribute may be queried using an
NV_CTRL_TARGET_TYPE_FRAMELOCK
 *    target type via XNVCTRLQueryTargetAttribute().
 *
 * X: When Xinerama is enabled, this attribute is kept consistent
across
 *    all Physical X Screens;  Assignment of this attribute will be
 *    broadcast by the NVIDIA X Driver to all X Screens.
 *
 * V: The attribute may be queried using an NV_CTRL_TARGET_TYPE_VCSC
 *    target type via XNVCTRLQueryTargetAttribute().
 *
 * I: The attribute may be queried using an NV_CTRL_TARGET_TYPE_GVI
target type
 *    via XNVCTRLQueryTargetAttribute().
 *
 * NOTE: Unless mentioned otherwise, all attributes may be queried
using
 *       an NV_CTRL_TARGET_TYPE_X_SCREEN target type via
 *       XNVCTRLQueryTargetAttribute().
 */


/**********************************************************************
****/

/*
 * Integer attributes:
 *
 * Integer attributes can be queried through the
XNVCTRLQueryAttribute() and
 * XNVCTRLQueryTargetAttribute() function calls.
 *
 * Integer attributes can be set through the XNVCTRLSetAttribute() and
 * XNVCTRLSetTargetAttribute() function calls.
 *
 * Unless otherwise noted, all integer attributes can be queried/set
 * using an NV_CTRL_TARGET_TYPE_X_SCREEN target.  Attributes that
cannot
 * take an NV_CTRL_TARGET_TYPE_X_SCREEN also cannot be queried/set
through
 * XNVCTRLQueryAttribute()/XNVCTRLSetAttribute() (Since these assume
 * an X Screen target).
 */



/**********************************************************************
****/
/*
 * The NV_CTRL_GVO_* integer attributes are used to configure GVO
 * (Graphics to Video Out).  This functionality is available, for
 * example, on the Quadro FX 4000 SDI graphics board.
 *
 * The following is a typical usage pattern for the GVO attributes:
 *
 * - query NV_CTRL_GVO_SUPPORTED to determine if the X screen supports
GV0.
 *
 * - specify NV_CTRL_GVO_SYNC_MODE (one of FREE_RUNNING, GENLOCK, or
 * FRAMELOCK); if you specify GENLOCK or FRAMELOCK, you should also
 * specify NV_CTRL_GVO_SYNC_SOURCE.
 *
 * - Use NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECTED and
 * NV_CTRL_GVO_SDI_SYNC_INPUT_DETECTED to detect what input syncs are
 * present.
 *
 * (If no analog sync is detected but it is known that a valid
 * bi-level or tri-level sync is connected set
 * NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECT_MODE appropriately and
 * retest with NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECTED).
 *
 * - if syncing to input sync, query the
 * NV_CTRL_GVIO_DETECTED_VIDEO_FORMAT attribute; note that Input video
 * format can only be queried after SYNC_SOURCE is specified.
 *
 * - specify the NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT
 *
 * - specify the NV_CTRL_GVO_DATA_FORMAT
 *
 * - specify any custom Color Space Conversion (CSC) matrix, offset,
 * and scale with XNVCTRLSetGvoColorConversion().
 *
 * - if using the GLX_NV_video_out extension to display one or more
 * pbuffers, call glXGetVideoDeviceNV() to lock the GVO output for use
 * by the GLX client; then bind the pbuffer(s) to the GVO output with
 * glXBindVideoImageNV() and send pbuffers to the GVO output with
 * glXSendPbufferToVideoNV(); see the GLX_NV_video_out spec for more
 * details.
 *
 * - if, rather than using the GLX_NV_video_out extension to display
 * GLX pbuffers on the GVO output, you wish display the X screen on
 * the GVO output, set NV_CTRL_GVO_DISPLAY_X_SCREEN to
 * NV_CTRL_GVO_DISPLAY_X_SCREEN_ENABLE.
 *
 * Note that setting most GVO attributes only causes the value to be
 * cached in the X server.  The values will be flushed to the hardware
 * either when NV_CTRL_GVO_DISPLAY_X_SCREEN is enabled, or when a GLX
 * pbuffer is bound to the GVO output (with glXBindVideoImageNV()).
 *
 * Note that GLX_NV_video_out and NV_CTRL_GVO_DISPLAY_X_SCREEN are
 * mutually exclusive.  If NV_CTRL_GVO_DISPLAY_X_SCREEN is enabled,
 * then glXGetVideoDeviceNV will fail.  Similarly, if a GLX client has
 * locked the GVO output (via glXGetVideoDeviceNV), then
 * NV_CTRL_GVO_DISPLAY_X_SCREEN will fail.  The NV_CTRL_GVO_GLX_LOCKED
 * event will be sent when a GLX client locks the GVO output.
 *
 */


/*
 * NV_CTRL_GVO_SUPPORTED - returns whether this X screen supports GVO;
 * if this screen does not support GVO output, then all other GVO
 * attributes are unavailable.
 */

#define NV_CTRL_GVO_SUPPORTED                                   67  /*
R-- */
#define NV_CTRL_GVO_SUPPORTED_FALSE                             0
#define NV_CTRL_GVO_SUPPORTED_TRUE                              1


/*
 * NV_CTRL_GVO_SYNC_MODE - selects the GVO sync mode; possible values
 * are:
 *
 * FREE_RUNNING - GVO does not sync to any external signal
 *
 * GENLOCK - the GVO output is genlocked to an incoming sync signal;
 * genlocking locks at hsync.  This requires that the output video
 * format exactly match the incoming sync video format.
 *
 * FRAMELOCK - the GVO output is frame locked to an incoming sync
 * signal; frame locking locks at vsync.  This requires that the output
 * video format have the same refresh rate as the incoming sync video
 * format.
 */

#define NV_CTRL_GVO_SYNC_MODE                                   68  /*
RW- */
#define NV_CTRL_GVO_SYNC_MODE_FREE_RUNNING                      0
#define NV_CTRL_GVO_SYNC_MODE_GENLOCK                           1
#define NV_CTRL_GVO_SYNC_MODE_FRAMELOCK                         2


/*
 * NV_CTRL_GVO_SYNC_SOURCE - if NV_CTRL_GVO_SYNC_MODE is set to either
 * GENLOCK or FRAMELOCK, this controls which sync source is used as
 * the incoming sync signal (either Composite or SDI).  If
 * NV_CTRL_GVO_SYNC_MODE is FREE_RUNNING, this attribute has no
 * effect.
 */

#define NV_CTRL_GVO_SYNC_SOURCE                                 69  /*
RW- */
#define NV_CTRL_GVO_SYNC_SOURCE_COMPOSITE                       0
#define NV_CTRL_GVO_SYNC_SOURCE_SDI                             1


/*
 * NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT - specifies the desired output
video
 * format for GVO devices or the desired input video format for GVI
devices.
 *
 * Note that for GVO, the valid video formats may vary depending on
 * the NV_CTRL_GVO_SYNC_MODE and the incoming sync video format.  See
 * the definition of NV_CTRL_GVO_SYNC_MODE.
 *
 * Note that when querying the ValidValues for this data type, the
 * values are reported as bits within a bitmask
 * (ATTRIBUTE_TYPE_INT_BITS); unfortunately, there are more valid
 * value bits than will fit in a single 32-bit value.  To solve this,
 * query the ValidValues for NV_CTRL_GVIO_OUTPUT_VIDEO_FORMAT to check
 * which of the first 31 VIDEO_FORMATS are valid, then query the
 * ValidValues for NV_CTRL_GVIO_OUTPUT_VIDEO_FORMAT2 to check which of
 * the VIDEO_FORMATS with value 32 and higher are valid.
 */

#define NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT                     70  /*
RW--I */

#define NV_CTRL_GVIO_VIDEO_FORMAT_NONE                          0
#define NV_CTRL_GVIO_VIDEO_FORMAT_487I_59_94_SMPTE259_NTSC      1
#define NV_CTRL_GVIO_VIDEO_FORMAT_576I_50_00_SMPTE259_PAL       2
#define NV_CTRL_GVIO_VIDEO_FORMAT_720P_59_94_SMPTE296           3
#define NV_CTRL_GVIO_VIDEO_FORMAT_720P_60_00_SMPTE296           4
```

```
#define NV_CTRL_GVIO_VIDEO_FORMAT_1035I_59_94_SMPTE260          5
#define NV_CTRL_GVIO_VIDEO_FORMAT_1035I_60_00_SMPTE260          6
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_50_00_SMPTE295          7
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_50_00_SMPTE274          8
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_59_94_SMPTE274          9
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_60_00_SMPTE274          10
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_23_976_SMPTE274         11
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_24_00_SMPTE274          12
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_25_00_SMPTE274          13
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_29_97_SMPTE274          14
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_30_00_SMPTE274          15
#define NV_CTRL_GVIO_VIDEO_FORMAT_720P_50_00_SMPTE296           16
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_48_00_SMPTE274          17
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_47_96_SMPTE274          18
#define NV_CTRL_GVIO_VIDEO_FORMAT_720P_30_00_SMPTE296           19
#define NV_CTRL_GVIO_VIDEO_FORMAT_720P_29_97_SMPTE296           20
#define NV_CTRL_GVIO_VIDEO_FORMAT_720P_25_00_SMPTE296           21
#define NV_CTRL_GVIO_VIDEO_FORMAT_720P_24_00_SMPTE296           22
#define NV_CTRL_GVIO_VIDEO_FORMAT_720P_23_98_SMPTE296           23
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080PSF_25_00_SMPTE274        24
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080PSF_29_97_SMPTE274        25
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080PSF_30_00_SMPTE274        26
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080PSF_24_00_SMPTE274        27
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080PSF_23_98_SMPTE274        28
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_30_00_SMPTE372          29
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_29_97_SMPTE372          30
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_60_00_SMPTE372          31
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_59_94_SMPTE372          32
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_25_00_SMPTE372          33
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_50_00_SMPTE372          34
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_24_00_SMPTE372          35
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_23_98_SMPTE372          36
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_48_00_SMPTE372          37
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_47_96_SMPTE372          38

#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_50_00_3G_LEVEL_A_SMPTE274  39
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_59_94_3G_LEVEL_A_SMPTE274  40
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_60_00_3G_LEVEL_A_SMPTE274  41
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_60_00_3G_LEVEL_B_SMPTE274  42
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_60_00_3G_LEVEL_B_SMPTE274  43
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_60_00_3G_LEVEL_B_SMPTE372  44
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_50_00_3G_LEVEL_B_SMPTE274  45
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_50_00_3G_LEVEL_B_SMPTE274  46
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_50_00_3G_LEVEL_B_SMPTE372  47
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_30_00_3G_LEVEL_B_SMPTE274  48
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_30_00_3G_LEVEL_B_SMPTE372  49
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_25_00_3G_LEVEL_B_SMPTE274  50
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_25_00_3G_LEVEL_B_SMPTE372  51
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_24_00_3G_LEVEL_B_SMPTE274  52
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_24_00_3G_LEVEL_B_SMPTE372  53
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_48_00_3G_LEVEL_B_SMPTE274  54
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_48_00_3G_LEVEL_B_SMPTE372  55
```

```
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_59_94_3G_LEVEL_B_SMPTE274  56
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_59_94_3G_LEVEL_B_SMPTE274  57
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_59_94_3G_LEVEL_B_SMPTE372  58
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_29_97_3G_LEVEL_B_SMPTE274  59
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_29_97_3G_LEVEL_B_SMPTE372  60
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080P_23_98_3G_LEVEL_B_SMPTE274  61
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048P_23_98_3G_LEVEL_B_SMPTE372  62
#define NV_CTRL_GVIO_VIDEO_FORMAT_1080I_47_96_3G_LEVEL_B_SMPTE274  63
#define NV_CTRL_GVIO_VIDEO_FORMAT_2048I_47_96_3G_LEVEL_B_SMPTE372  64
/*


/*
 * The following are deprecated; NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT
and the
 * corresponding NV_CTRL_GVIO_* formats should be used instead.
 */
#define NV_CTRL_GVO_OUTPUT_VIDEO_FORMAT                         70  /*
RW- */

#define NV_CTRL_GVO_VIDEO_FORMAT_NONE                           0
#define NV_CTRL_GVO_VIDEO_FORMAT_487I_59_94_SMPTE259_NTSC       1
#define NV_CTRL_GVO_VIDEO_FORMAT_576I_50_00_SMPTE259_PAL        2
#define NV_CTRL_GVO_VIDEO_FORMAT_720P_59_94_SMPTE296            3
#define NV_CTRL_GVO_VIDEO_FORMAT_720P_60_00_SMPTE296            4
#define NV_CTRL_GVO_VIDEO_FORMAT_1035I_59_94_SMPTE260           5
#define NV_CTRL_GVO_VIDEO_FORMAT_1035I_60_00_SMPTE260           6
#define NV_CTRL_GVO_VIDEO_FORMAT_1080I_50_00_SMPTE295           7
#define NV_CTRL_GVO_VIDEO_FORMAT_1080I_50_00_SMPTE274           8
#define NV_CTRL_GVO_VIDEO_FORMAT_1080I_59_94_SMPTE274           9
#define NV_CTRL_GVO_VIDEO_FORMAT_1080I_60_00_SMPTE274           10
#define NV_CTRL_GVO_VIDEO_FORMAT_1080P_23_976_SMPTE274          11
#define NV_CTRL_GVO_VIDEO_FORMAT_1080P_24_00_SMPTE274           12
#define NV_CTRL_GVO_VIDEO_FORMAT_1080P_25_00_SMPTE274           13
#define NV_CTRL_GVO_VIDEO_FORMAT_1080P_29_97_SMPTE274           14
#define NV_CTRL_GVO_VIDEO_FORMAT_1080P_30_00_SMPTE274           15
#define NV_CTRL_GVO_VIDEO_FORMAT_720P_50_00_SMPTE296            16
#define NV_CTRL_GVO_VIDEO_FORMAT_1080I_48_00_SMPTE274           17
#define NV_CTRL_GVO_VIDEO_FORMAT_1080I_47_96_SMPTE274           18
#define NV_CTRL_GVO_VIDEO_FORMAT_720P_30_00_SMPTE296            19
#define NV_CTRL_GVO_VIDEO_FORMAT_720P_29_97_SMPTE296            20
#define NV_CTRL_GVO_VIDEO_FORMAT_720P_25_00_SMPTE296            21
#define NV_CTRL_GVO_VIDEO_FORMAT_720P_24_00_SMPTE296            22
#define NV_CTRL_GVO_VIDEO_FORMAT_720P_23_98_SMPTE296            23
#define NV_CTRL_GVO_VIDEO_FORMAT_1080PSF_25_00_SMPTE274         24
#define NV_CTRL_GVO_VIDEO_FORMAT_1080PSF_29_97_SMPTE274         25
#define NV_CTRL_GVO_VIDEO_FORMAT_1080PSF_30_00_SMPTE274         26
#define NV_CTRL_GVO_VIDEO_FORMAT_1080PSF_24_00_SMPTE274         27
#define NV_CTRL_GVO_VIDEO_FORMAT_1080PSF_23_98_SMPTE274         28
#define NV_CTRL_GVO_VIDEO_FORMAT_2048P_30_00_SMPTE372           29
#define NV_CTRL_GVO_VIDEO_FORMAT_2048P_29_97_SMPTE372           30
#define NV_CTRL_GVO_VIDEO_FORMAT_2048I_60_00_SMPTE372           31
```

```
#define NV_CTRL_GVO_VIDEO_FORMAT_2048I_59_94_SMPTE372           32
#define NV_CTRL_GVO_VIDEO_FORMAT_2048P_25_00_SMPTE372           33
#define NV_CTRL_GVO_VIDEO_FORMAT_2048I_50_00_SMPTE372           34
#define NV_CTRL_GVO_VIDEO_FORMAT_2048P_24_00_SMPTE372           35
#define NV_CTRL_GVO_VIDEO_FORMAT_2048P_23_98_SMPTE372           36
#define NV_CTRL_GVO_VIDEO_FORMAT_2048I_48_00_SMPTE372           37
#define NV_CTRL_GVO_VIDEO_FORMAT_2048I_47_96_SMPTE372           38

/*
 * NV_CTRL_GVIO_DETECTED_VIDEO_FORMAT - indicates the input video
format
 * detected for GVO or GVI devices; the possible values are the
 * NV_CTRL_GVIO_VIDEO_FORMAT constants.
 *
 * For GVI devices, the jack number should be specified in the lower
 * 16 bits of the "display_mask" parameter, while the channel number
should be
 * specified in the upper 16 bits.
 */

#define NV_CTRL_GVIO_DETECTED_VIDEO_FORMAT                      71  /*
R--I */

/*
 * The following is deprecated.  Use
NV_CTRL_GVIO_DETECTED_VIDEO_FORMAT,
 * instead.
 */
#define NV_CTRL_GVO_INPUT_VIDEO_FORMAT                          71  /*
R-- */

/*
 * NV_CTRL_GVO_DATA_FORMAT - This controls how the data in the source
 * (either the X screen or the GLX pbuffer) is interpretted and
 * displayed.
 *
 * Note: some of the below DATA_FORMATS have been renamed.  For
 * example, R8G8B8_TO_RGB444 has been renamed to X8X8X8_444_PASSTHRU.
 * This is to more accurately reflect DATA_FORMATS where the
 * per-channel data could be either RGB or YCrCb -- the point is that
 * the driver and GVO hardware do not perform any implicit color space
 * conversion on the data; it is passed through to the SDI out.
 */

#define NV_CTRL_GVO_DATA_FORMAT                                 72  /*
RW- */
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8_TO_YCRCB444              0
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8A8_TO_YCRCBA4444          1
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8Z10_TO_YCRCBZ4444         2
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8_TO_YCRCB422              3
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8A8_TO_YCRCBA4224          4
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8Z10_TO_YCRCBZ4224         5
```

```
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8_TO_RGB444                6 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_X8X8X8_444_PASSTHRU             6
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8A8_TO_RGBA4444            7 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_X8X8X8A8_4444_PASSTHRU          7
#define NV_CTRL_GVO_DATA_FORMAT_R8G8B8Z10_TO_RGBZ4444           8 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_X8X8X8Z8_4444_PASSTHRU          8
#define NV_CTRL_GVO_DATA_FORMAT_Y10CR10CB10_TO_YCRCB444         9 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_X10X10X10_444_PASSTHRU          9
#define NV_CTRL_GVO_DATA_FORMAT_Y10CR8CB8_TO_YCRCB444           10 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_X10X8X8_444_PASSTHRU            10
#define NV_CTRL_GVO_DATA_FORMAT_Y10CR8CB8A10_TO_YCRCBA4444      11 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_X10X8X8A10_4444_PASSTHRU        11
#define NV_CTRL_GVO_DATA_FORMAT_Y10CR8CB8Z10_TO_YCRCBZ4444      12 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_X10X8X8Z10_4444_PASSTHRU        12
#define NV_CTRL_GVO_DATA_FORMAT_DUAL_R8G8B8_TO_DUAL_YCRCB422    13
#define NV_CTRL_GVO_DATA_FORMAT_DUAL_Y8CR8CB8_TO_DUAL_YCRCB422  14 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_DUAL_X8X8X8_TO_DUAL_422_PASSTHRU 14
#define NV_CTRL_GVO_DATA_FORMAT_R10G10B10_TO_YCRCB422           15
#define NV_CTRL_GVO_DATA_FORMAT_R10G10B10_TO_YCRCB444           16
#define NV_CTRL_GVO_DATA_FORMAT_Y12CR12CB12_TO_YCRCB444         17 //
renamed
#define NV_CTRL_GVO_DATA_FORMAT_X12X12X12_444_PASSTHRU          17
#define NV_CTRL_GVO_DATA_FORMAT_R12G12B12_TO_YCRCB444           18
#define NV_CTRL_GVO_DATA_FORMAT_X8X8X8_422_PASSTHRU             19
#define NV_CTRL_GVO_DATA_FORMAT_X8X8X8A8_4224_PASSTHRU          20
#define NV_CTRL_GVO_DATA_FORMAT_X8X8X8Z8_4224_PASSTHRU          21
#define NV_CTRL_GVO_DATA_FORMAT_X10X10X10_422_PASSTHRU          22
#define NV_CTRL_GVO_DATA_FORMAT_X10X8X8_422_PASSTHRU            23
#define NV_CTRL_GVO_DATA_FORMAT_X10X8X8A10_4224_PASSTHRU        24
#define NV_CTRL_GVO_DATA_FORMAT_X10X8X8Z10_4224_PASSTHRU        25
#define NV_CTRL_GVO_DATA_FORMAT_X12X12X12_422_PASSTHRU          26
#define NV_CTRL_GVO_DATA_FORMAT_R12G12B12_TO_YCRCB422           27

/*
 * NV_CTRL_GVO_DISPLAY_X_SCREEN - enable/disable GVO output of the X
 * screen (in Clone mode).  At this point, all the GVO attributes that
 * have been cached in the X server are flushed to the hardware and GVO
is
 * enabled.  Note that this attribute can fail to be set if a GLX
 * client has locked the GVO output (via glXGetVideoDeviceNV).  Note
 * that due to the inherit race conditions in this locking strategy,
 * NV_CTRL_GVO_DISPLAY_X_SCREEN can fail unexpectantly.  In the
 * failing situation, X will not return an X error.  Instead, you
 * should query the value of NV_CTRL_GVO_DISPLAY_X_SCREEN after
 * setting it to confirm that the setting was applied.
 *
 * NOTE: This attribute is related to the NV_CTRL_GVO_LOCK_OWNER
 *       attribute.  When NV_CTRL_GVO_DISPLAY_X_SCREEN is enabled,
 *       the GVO device will be locked by NV_CTRL_GVO_LOCK_OWNER_CLONE.
 *       see NV_CTRL_GVO_LOCK_OWNER for detais.
 */

#define NV_CTRL_GVO_DISPLAY_X_SCREEN                            73  /*
RW- */
#define NV_CTRL_GVO_DISPLAY_X_SCREEN_ENABLE                     1
#define NV_CTRL_GVO_DISPLAY_X_SCREEN_DISABLE                    0


/*
 * NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECTED - indicates whether
 * Composite Sync input is detected.
 */

#define NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECTED               74  /*
R-- */
#define NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECTED_FALSE         0
#define NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECTED_TRUE          1


/*
 * NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECT_MODE - get/set the
 * Composite Sync input detect mode.
 */

#define NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECT_MODE            75  /*
RW- */
#define NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECT_MODE_AUTO       0
#define NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECT_MODE_BI_LEVEL   1
#define NV_CTRL_GVO_COMPOSITE_SYNC_INPUT_DETECT_MODE_TRI_LEVEL  2


/*
 * NV_CTRL_GVO_SYNC_INPUT_DETECTED - indicates whether SDI Sync input
 * is detected, and what type.
 */

#define NV_CTRL_GVO_SDI_SYNC_INPUT_DETECTED                     76  /*
R-- */
#define NV_CTRL_GVO_SDI_SYNC_INPUT_DETECTED_NONE                0
#define NV_CTRL_GVO_SDI_SYNC_INPUT_DETECTED_HD                  1
#define NV_CTRL_GVO_SDI_SYNC_INPUT_DETECTED_SD                  2


/*
 * NV_CTRL_GVO_VIDEO_OUTPUTS - indicates which GVO video output
 * connectors are currently outputing data.
 */
```

```
#define NV_CTRL_GVO_VIDEO_OUTPUTS                               77  /*
R-- */
#define NV_CTRL_GVO_VIDEO_OUTPUTS_NONE                          0
#define NV_CTRL_GVO_VIDEO_OUTPUTS_VIDEO1                        1
#define NV_CTRL_GVO_VIDEO_OUTPUTS_VIDEO2                        2
#define NV_CTRL_GVO_VIDEO_OUTPUTS_VIDEO_BOTH                    3


/*
 * NV_CTRL_GVO_FPGA_VERSION - indicates the version of the Firmware on
 * the GVO device.  Deprecated; use
 * NV_CTRL_STRING_GVIO_FIRMWARE_VERSION instead.
 */

#define NV_CTRL_GVO_FIRMWARE_VERSION                            78  /*
R-- */


/*
 * NV_CTRL_GVO_SYNC_DELAY_PIXELS - controls the delay between the
 * input sync and the output sync in numbers of pixels from hsync;
 * this is a 12 bit value.
 *
 * If the NV_CTRL_GVO_CAPABILITIES_ADVANCE_SYNC_SKEW bit is set,
 * then setting this value will set an advance instead of a delay.
 */

#define NV_CTRL_GVO_SYNC_DELAY_PIXELS                           79  /*
RW- */


/*
 * NV_CTRL_GVO_SYNC_DELAY_LINES - controls the delay between the input
 * sync and the output sync in numbers of lines from vsync; this is a
 * 12 bit value.
 *
 * If the NV_CTRL_GVO_CAPABILITIES_ADVANCE_SYNC_SKEW bit is set,
 * then setting this value will set an advance instead of a delay.
 */

#define NV_CTRL_GVO_SYNC_DELAY_LINES                            80  /*
RW- */


/*
 * NV_CTRL_GVO_INPUT_VIDEO_FORMAT_REACQUIRE - must be set for a period
 * of about 2 seconds for the new InputVideoFormat to be properly
 * locked to.  In nvidia-settings, we do a reacquire whenever genlock
 * or frame lock mode is entered into, when the user clicks the
 * "detect" button.  This value can be written, but always reads back
 * _FALSE.
 */

#define NV_CTRL_GVO_INPUT_VIDEO_FORMAT_REACQUIRE                81  /*
-W- */
#define NV_CTRL_GVO_INPUT_VIDEO_FORMAT_REACQUIRE_FALSE          0
#define NV_CTRL_GVO_INPUT_VIDEO_FORMAT_REACQUIRE_TRUE           1


/*
 * NV_CTRL_GVO_GLX_LOCKED - indicates that GVO configurability is
locked by
 * GLX;  this occurs when the GLX_NV_video_out function calls
 * glXGetVideoDeviceNV().  All GVO output resources are locked until
 * either glXReleaseVideoDeviceNV() is called or the X Display used
 * when calling glXGetVideoDeviceNV() is closed.
 *
 * When GVO is locked, setting of the following GVO NV-CONTROL
attributes will
 * not happen immediately and will instead be cached.  The GVO resource
will
 * need to be disabled/released and re-enabled/claimed for the values
to be
 * flushed. These attributes are:
 *    NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT
 *    NV_CTRL_GVO_DATA_FORMAT
 *    NV_CTRL_GVO_FLIP_QUEUE_SIZE
 *
 * XXX This is deprecated, please see NV_CTRL_GVO_LOCK_OWNER
 */

#define NV_CTRL_GVO_GLX_LOCKED                                  82  /*
R-- */
#define NV_CTRL_GVO_GLX_LOCKED_FALSE                            0
#define NV_CTRL_GVO_GLX_LOCKED_TRUE                             1


/*
 * NV_CTRL_GVIO_VIDEO_FORMAT_{WIDTH,HEIGHT,REFRESH_RATE} - query the
 * width, height, and refresh rate for the specified
 * NV_CTRL_GVIO_VIDEO_FORMAT_*.  So that this can be queried with
 * existing interfaces, XNVCTRLQueryAttribute() should be used, and
 * the video format specified in the display_mask field; eg:
 *
 * XNVCTRLQueryAttribute (dpy,
 *                        screen,
 *
NV_CTRL_GVIO_VIDEO_FORMAT_487I_59_94_SMPTE259_NTSC,
 *                        NV_CTRL_GVIO_VIDEO_FORMAT_WIDTH,
 *                        &value);
 *
 * Note that Refresh Rate is in milliHertz values
 */
```

```
#define NV_CTRL_GVIO_VIDEO_FORMAT_WIDTH                         83  /*
R--I */
#define NV_CTRL_GVIO_VIDEO_FORMAT_HEIGHT                        84  /*
R--I */
#define NV_CTRL_GVIO_VIDEO_FORMAT_REFRESH_RATE                  85  /*
R--I */

/* The following are deprecated; use the NV_CTRL_GVIO_* versions,
instead */
#define NV_CTRL_GVO_VIDEO_FORMAT_WIDTH                          83  /*
R-- */
#define NV_CTRL_GVO_VIDEO_FORMAT_HEIGHT                         84  /*
R-- */
#define NV_CTRL_GVO_VIDEO_FORMAT_REFRESH_RATE                   85  /*
R-- */


/*
 * NV_CTRL_GVO_X_SCREEN_PAN_[XY] - when GVO output of the X screen is
 * enabled, the pan x/y attributes control which portion of the X
 * screen is displayed by GVO.  These attributes can be updated while
 * GVO output is enabled, or before enabling GVO output.  The pan
 * values will be clamped so that GVO output is not panned beyond the
 * end of the X screen.
 */

#define NV_CTRL_GVO_X_SCREEN_PAN_X                              86  /*
RW- */
#define NV_CTRL_GVO_X_SCREEN_PAN_Y                              87  /*
RW- */

/*
 * NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT2 - this attribute is only
intended
 * to be used to query the ValidValues for
 * NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT above the first 31
VIDEO_FORMATS.
 * See NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT for details.
 */

#define NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT2                    227  /*
---GI */

/*
 * The following is deprecated; use
NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT2,
 * instead
 */
#define NV_CTRL_GVO_OUTPUT_VIDEO_FORMAT2                         227
/* --- */


/*
```

```
 * NV_CTRL_GVO_OVERRIDE_HW_CSC - Override the SDI hardware's Color
Space
 * Conversion with the values controlled through
 * XNVCTRLSetGvoColorConversion() and XNVCTRLGetGvoColorConversion().
If
 * this attribute is FALSE, then the values specified through
 * XNVCTRLSetGvoColorConversion() are ignored.
 */

#define NV_CTRL_GVO_OVERRIDE_HW_CSC                              228
/* RW- */
#define NV_CTRL_GVO_OVERRIDE_HW_CSC_FALSE                        0
#define NV_CTRL_GVO_OVERRIDE_HW_CSC_TRUE                         1


/*
 * NV_CTRL_GVO_CAPABILITIES - this read-only attribute describes GVO
 * capabilities that differ between NVIDIA SDI products.  This value
 * is a bitmask where each bit indicates whether that capability is
 * available.
 *
 * APPLY_CSC_IMMEDIATELY - whether the CSC matrix, offset, and scale
 * specified through XNVCTRLSetGvoColorConversion() will take affect
 * immediately, or only after SDI output is disabled and enabled
 * again.
 *
 * APPLY_CSC_TO_X_SCREEN - whether the CSC matrix, offset, and scale
 * specified through XNVCTRLSetGvoColorConversion() will also apply
 * to GVO output of an X screen, or only to OpenGL GVO output, as
 * enabled through the GLX_NV_video_out extension.
 *
 * COMPOSITE_TERMINATION - whether the 75 ohm termination of the
 * SDI composite input signal can be programmed through the
 * NV_CTRL_GVO_COMPOSITE_TERMINATION attribute.
 *
 * SHARED_SYNC_BNC - whether the SDI device has a single BNC
 * connector used for both (SDI & Composite) incoming signals.
 *
 * MULTIRATE_SYNC - whether the SDI device supports synchronization
 * of input and output video modes that match in being odd or even
 * modes (ie, AA.00 Hz modes can be synched to other BB.00 Hz modes and
 * AA.XX Hz can match to BB.YY Hz where .XX and .YY are not .00)
 */

#define NV_CTRL_GVO_CAPABILITIES                                 229
/* R-- */
#define NV_CTRL_GVO_CAPABILITIES_APPLY_CSC_IMMEDIATELY
0x00000001
#define NV_CTRL_GVO_CAPABILITIES_APPLY_CSC_TO_X_SCREEN
0x00000002
#define NV_CTRL_GVO_CAPABILITIES_COMPOSITE_TERMINATION
0x00000004
```

```
#define NV_CTRL_GVO_CAPABILITIES_SHARED_SYNC_BNC
0x00000008
#define NV_CTRL_GVO_CAPABILITIES_MULTIRATE_SYNC
0x00000010
#define NV_CTRL_GVO_CAPABILITIES_ADVANCE_SYNC_SKEW
0x00000020


/*
 * NV_CTRL_GVO_COMPOSITE_TERMINATION - enable or disable 75 ohm
 * termination of the SDI composite input signal.
 */

#define NV_CTRL_GVO_COMPOSITE_TERMINATION                        230
/* RW- */
#define NV_CTRL_GVO_COMPOSITE_TERMINATION_ENABLE                   1
#define NV_CTRL_GVO_COMPOSITE_TERMINATION_DISABLE                  0

/*
 * NV_CTRL_GVO_FLIP_QUEUE_SIZE - The Graphics to Video Out interface
 * exposed through NV-CONTROL and the GLX_NV_video_out extension uses
 * an internal flip queue when pbuffers are sent to the video device
 * (via glXSendPbufferToVideoNV()).  The NV_CTRL_GVO_FLIP_QUEUE_SIZE
 * can be used to query and assign the flip queue size.  This
 * attribute is applied to GLX when glXGetVideoDeviceNV() is called by
 * the application.
 */

#define NV_CTRL_GVO_FLIP_QUEUE_SIZE                              236 /*
RW- */

/*
 * NV_CTRL_GVO_LOCK_OWNER - indicates that the GVO device is available
 * or in use (by GLX, Clone Mode, TwinView etc).
 *
 * The GVO device is locked by GLX when the GLX_NV_video_out function
 * calls glXGetVideoDeviceNV().  The GVO device is then unlocked when
 * glXReleaseVideoDeviceNV() is called, or the X Display used when
calling
 * glXGetVideoDeviceNV() is closed.
 *
 * The GVO device is locked/unlocked for Clone mode use when the
 * attribute NV_CTRL_GVO_DISPLAY_X_SCREEN is enabled/disabled.
 *
 * The GVO device is locked/unlocked by TwinView mode, when the GVO
device is
 * associated/unassociated to/from an X screen through the
 * NV_CTRL_ASSOCIATED_DISPLAY_DEVICES attribute directly.
 *
 * When the GVO device is locked, setting of the following GVO NV-
CONTROL
 * attributes will not happen immediately and will instead be cached.
The
```

```
 * GVO resource will need to be disabled/released and re-
enabled/claimed for
 * the values to be flushed. These attributes are:
 *
 *    NV_CTRL_GVIO_REQUESTED_VIDEO_FORMAT
 *    NV_CTRL_GVO_DATA_FORMAT
 *    NV_CTRL_GVO_FLIP_QUEUE_SIZE
 */

#define NV_CTRL_GVO_LOCK_OWNER                                  257 /*
R-- */
#define NV_CTRL_GVO_LOCK_OWNER_NONE                               0
#define NV_CTRL_GVO_LOCK_OWNER_GLX                                1
#define NV_CTRL_GVO_LOCK_OWNER_CLONE                              2
#define NV_CTRL_GVO_LOCK_OWNER_X_SCREEN                           3
/*
 * NV_CTRL_GVO_OUTPUT_VIDEO_LOCKED - Returns whether or not the GVO
output
 * video is locked to the GPU.
 */

#define NV_CTRL_GVO_OUTPUT_VIDEO_LOCKED                         267 /*
R--- */
#define NV_CTRL_GVO_OUTPUT_VIDEO_LOCKED_FALSE                     0
#define NV_CTRL_GVO_OUTPUT_VIDEO_LOCKED_TRUE                      1


/*
 * NV_CTRL_GVO_SYNC_LOCK_STATUS - Returns whether or not the GVO device
 * is locked to the input ref signal.  If the sync mode is set to
 * NV_CTRL_GVO_SYNC_MODE_GENLOCK, then this returns the genlock
 * sync status, and if the sync mode is set to
NV_CTRL_GVO_SYNC_MODE_FRAMELOCK,
 * then this reports the frame lock status.
 */

#define NV_CTRL_GVO_SYNC_LOCK_STATUS                            268 /*
R--- */
#define NV_CTRL_GVO_SYNC_LOCK_STATUS_UNLOCKED                     0
#define NV_CTRL_GVO_SYNC_LOCK_STATUS_LOCKED                       1


/*
 * NV_CTRL_GVO_ANC_TIME_CODE_GENERATION - Allows SDI device to generate
 * time codes in the ANC region of the SDI video output stream.
 */

#define NV_CTRL_GVO_ANC_TIME_CODE_GENERATION                    269 /*
RW-- */
#define NV_CTRL_GVO_ANC_TIME_CODE_GENERATION_DISABLE              0
#define NV_CTRL_GVO_ANC_TIME_CODE_GENERATION_ENABLE               1
```

```
/*
 * NV_CTRL_GVO_COMPOSITE - Enables/Disables SDI compositing.  This
attribute
 * is only available when an SDI input source is detected and is in
genlock
 * mode.
 */

#define NV_CTRL_GVO_COMPOSITE                                   270 /*
RW-- */
#define NV_CTRL_GVO_COMPOSITE_DISABLE                             0
#define NV_CTRL_GVO_COMPOSITE_ENABLE                              1


/*
 * NV_CTRL_GVO_COMPOSITE_ALPHA_KEY - When compositing is enabled, this
 * enables/disables alpha blending.
 */

#define NV_CTRL_GVO_COMPOSITE_ALPHA_KEY                         271 /*
RW-- */
#define NV_CTRL_GVO_COMPOSITE_ALPHA_KEY_DISABLE                   0
#define NV_CTRL_GVO_COMPOSITE_ALPHA_KEY_ENABLE                    1


/*
 * NV_CTRL_GVO_COMPOSITE_LUMA_KEY_RANGE - Set the values of a luma
 * channel range.  This is a packed int that has the following format
 * (in order of high-bits to low bits):
 *
 * Range # (11 bits), (Enabled 1 bit), min value (10 bits), max value
(10 bits)
 *
 * To query the current values, pass the range # throught he
display_mask
 * variable.
 */

#define NV_CTRL_GVO_COMPOSITE_LUMA_KEY_RANGE                    272 /*
RW-- */

#define NV_CTRL_GVO_COMPOSITE_MAKE_RANGE(range, enable, min, max) \
    ((((min) & 0x3FF)   <<  0) |  \
     (((max) & 0x3FF)   << 10) |  \
     (((enable) & 0x1)  << 20) |  \
     (((range) & 0x7FF) << 21))

#define NV_CTRL_GVO_COMPOSITE_GET_RANGE(val, range, enable, min, max) \
    (min)    = ((val) >> 0)  & 0x3FF; \
    (max)    = ((val) >> 10) & 0x3FF; \
    (enable) = ((val) >> 20) & 0x1;   \
    (range)  = ((val) >> 21) & 0x7FF;
```

```
/*
 * NV_CTRL_GVO_COMPOSITE_CR_KEY_RANGE - Set the values of a CR
 * channel range.  This is a packed int that has the following format
 * (in order of high-bits to low bits):
 *
 * Range # (11 bits), (Enabled 1 bit), min value (10 bits), max value
(10 bits)
 *
 * To query the current values, pass the range # throught he
display_mask
 * variable.
 */

#define NV_CTRL_GVO_COMPOSITE_CR_KEY_RANGE                      273 /*
RW-- */


/*
 * NV_CTRL_GVO_COMPOSITE_CB_KEY_RANGE - Set the values of a CB
 * channel range.  This is a packed int that has the following format
 * (in order of high-bits to low bits):
 *
 * Range # (11 bits), (Enabled 1 bit), min value (10 bits), max value
(10 bits)
 *
 * To query the current values, pass the range # throught he
display_mask
 * variable.
 */

#define NV_CTRL_GVO_COMPOSITE_CB_KEY_RANGE                      274 /*
RW-- */


/*
 * NV_CTRL_GVO_COMPOSITE_NUM_KEY_RANGES - Returns the number of ranges
 * available for each channel (Y/Luma, Cr, and Cb.)
 */

#define NV_CTRL_GVO_COMPOSITE_NUM_KEY_RANGES                    275 /*
R--- */

/*
 * NV_CTRL_GVO_CSC_CHANGED_EVENT This attribute is sent as an event
 * when the color space conversion matrix has been altered by another
 * client.
 */

#define NV_CTRL_GVO_CSC_CHANGED_EVENT

/*
 * NV_CTRL_GVO_SYNC_TO_DISPLAY This attribute controls whether or not
 * the non-SDI display device will be sync'ed to the SDI display device
 * (when configured in TwinView, Clone Mode or when using the SDI
device
 * with OpenGL).
 */

#define NV_CTRL_GVO_SYNC_TO_DISPLAY                             296 /*
--- */
#define NV_CTRL_GVO_SYNC_TO_DISPLAY_DISABLE                     0
#define NV_CTRL_GVO_SYNC_TO_DISPLAY_ENABLE                      1

/*
 * NV_CTRL_IS_GVO_DISPLAY - returns whether or not a given display is
an
 * SDI device.
 */

#define NV_CTRL_IS_GVO_DISPLAY                                  300 /*
R-D */
#define NV_CTRL_IS_GVO_DISPLAY_FALSE                              0
#define NV_CTRL_IS_GVO_DISPLAY_TRUE                               1

/*
 * NV_CTRL_GVO_FULL_RANGE_COLOR - Allow full range color data [4-1019]
 * without clamping to [64-940].
 */

#define NV_CTRL_GVO_FULL_RANGE_COLOR                            302 /*
RW- */
#define NV_CTRL_GVO_FULL_RANGE_COLOR_DISABLED                     0
#define NV_CTRL_GVO_FULL_RANGE_COLOR_ENABLED                      1
/*
 * NV_CTRL_GVO_ENABLE_RGB_DATA - Allows clients to specify when
 * the GVO board should process colors as RGB when the output data
 * format is one of the NV_CTRL_GVO_DATA_FORMAT_???_PASSTRHU modes.
 */

#define NV_CTRL_GVO_ENABLE_RGB_DATA                             304 /*
RW- */
#define NV_CTRL_GVO_ENABLE_RGB_DATA_DISABLE                       0
#define NV_CTRL_GVO_ENABLE_RGB_DATA_ENABLE                        1

/*
 * NV_CTRL_GVI_NUM_JACKS - Returns the number of input BNC jacks
available
 * on a GVI device.
 */

#define NV_CTRL_GVI_NUM_JACKS                                   307 /*
R--I */

/*
```

```
 * NV_CTRL_GVI_MAX_LINKS_PER_STREAM - Returns the maximum supported
number of
 * links that can be tied to one stream.
 */

#define NV_CTRL_GVI_MAX_LINKS_PER_STREAM                        308 /*
R--I */

/*
 * NV_CTRL_GVI_DETECTED_CHANNEL_BITS_PER_COMPONENT - Returns the
detected
 * number of bits per component (BPC) of data on the given input jack+
 * channel.
 *
 * The jack number should be specified in the lower 16 bits of the
 * "display_mask" parameter, while the channel number should be
specified in
 * the upper 16 bits.
 */

#define NV_CTRL_GVI_DETECTED_CHANNEL_BITS_PER_COMPONENT         309 /*
R--I */
#define NV_CTRL_GVI_BITS_PER_COMPONENT_UNKNOWN                    0
#define NV_CTRL_GVI_BITS_PER_COMPONENT_8                          1
#define NV_CTRL_GVI_BITS_PER_COMPONENT_10                         2
#define NV_CTRL_GVI_BITS_PER_COMPONENT_12                         3

/*
 * NV_CTRL_GVI_REQUESTED_STREAM_BITS_PER_COMPONENT - Specify the number
of
 * bits per component (BPC) of data for the captured stream.
 * The stream number should be specified in the "display_mask"
parameter.
 */

#define NV_CTRL_GVI_REQUESTED_STREAM_BITS_PER_COMPONENT         310 /*
RW-I */

/*
 * NV_CTRL_GVI_DETECTED_CHANNEL_COMPONENT_SAMPLING - Returns the
detected
 * sampling format for the input jack+channel.
 *
 * The jack number should be specified in the lower 16 bits of the
 * "display_mask" parameter, while the channel number should be
specified in
 * the upper 16 bits.
 */

#define NV_CTRL_GVI_DETECTED_CHANNEL_COMPONENT_SAMPLING         311 /*
R--I */
#define NV_CTRL_GVI_COMPONENT_SAMPLING_UNKNOWN                    0
#define NV_CTRL_GVI_COMPONENT_SAMPLING_4444                       1
```

```
#define NV_CTRL_GVI_COMPONENT_SAMPLING_4224                       2
#define NV_CTRL_GVI_COMPONENT_SAMPLING_444                        3
#define NV_CTRL_GVI_COMPONENT_SAMPLING_422                        4
#define NV_CTRL_GVI_COMPONENT_SAMPLING_420                        5

/*
 * NV_CTRL_GVI_REQUESTED_COMPONENT_SAMPLING - Specify the sampling
format for
 * the captured stream.
 * The possible values are the NV_CTRL_GVI_DETECTED_COMPONENT_SAMPLING
 * constants.
 * The stream number should be specified in the "display_mask"
parameter.
 */

#define NV_CTRL_GVI_REQUESTED_STREAM_COMPONENT_SAMPLING         312 /*
RW-I */

/*
 * NV_CTRL_GVI_CHROMA_EXPAND - Enable or disable 4:2:2 -> 4:4:4 chroma
 * expansion for the captured stream.  This value is ignored when a
 * COMPONENT_SAMPLING format is selected that does not use chroma
subsampling.
 * The stream number should be specified in the "display_mask"
parameter.
 */

#define NV_CTRL_GVI_REQUESTED_STREAM_CHROMA_EXPAND              313 /*
RW-I */
#define NV_CTRL_GVI_CHROMA_EXPAND_FALSE                           0
#define NV_CTRL_GVI_CHROMA_EXPAND_TRUE                            1

/*
 * NV_CTRL_GVI_DETECTED_CHANNEL_COLOR_SPACE - Returns the detected
color space
 * of the input jack+channel.
 *
 * The jack number should be specified in the lower 16 bits of the
 * "display_mask" parameter, while the channel number should be
specified in
 * the upper 16 bits.
 */

#define NV_CTRL_GVI_DETECTED_CHANNEL_COLOR_SPACE                314 /*
R--I */
#define NV_CTRL_GVI_COLOR_SPACE_UNKNOWN                           0
#define NV_CTRL_GVI_COLOR_SPACE_GBR                               1
#define NV_CTRL_GVI_COLOR_SPACE_GBRA                              2
#define NV_CTRL_GVI_COLOR_SPACE_GBRD                              3
#define NV_CTRL_GVI_COLOR_SPACE_YCBCR                             4
#define NV_CTRL_GVI_COLOR_SPACE_YCBCRA                            5
#define NV_CTRL_GVI_COLOR_SPACE_YCBCRD                            6
```

```
/*
 * NV_CTRL_GVI_DETECTED_CHANNEL_LINK_ID - Returns the detected link
identifier
 * for the given input jack+channel.
 *
 * The jack number should be specified in the lower 16 bits of the
 * "display_mask" parameter, while the channel number should be
specified in
 * the upper 16 bits.
 */

#define NV_CTRL_GVI_DETECTED_CHANNEL_LINK_ID                    315 /*
R--I */
#define NV_CTRL_GVI_LINK_ID_UNKNOWN                          0xFFFF

/*
 * NV_CTRL_GVI_DETECTED_CHANNEL_SMPTE352_IDENTIFIER - Returns the 4-
byte
 * SMPTE 352 identifier from the given input jack+channel.
 *
 * The jack number should be specified in the lower 16 bits of the
 * "display_mask" parameter, while the channel number should be
specified in
 * the upper 16 bits.
 */

#define NV_CTRL_GVI_DETECTED_CHANNEL_SMPTE352_IDENTIFIER        316 /*
R--I */

/*
 * NV_CTRL_GVI_GLOBAL_IDENTIFIER - Returns a global identifier for the
 * GVI device.  This identifier can be used to relate GVI devices named
 * in NV-CONTROL with those enumerated in OpenGL.
 */

#define NV_CTRL_GVI_GLOBAL_IDENTIFIER                           317 /*
R--I */

/*
 * NV_CTRL_FRAMELOCK_SYNC_DELAY_RESOLUTION - Returns the number of
nanoseconds
 * that one unit of NV_CTRL_FRAMELOCK_SYNC_DELAY corresponds to.
 */
#define NV_CTRL_FRAMELOCK_SYNC_DELAY_RESOLUTION                 318 /*
R-- */

/*
 * NV_CTRL_GVI_SYNC_OUTPUT_FORMAT - Returns the output sync signal
 * from the GVI device.
 */

#define NV_CTRL_GVI_SYNC_OUTPUT_FORMAT                          335 /*
R--I */
```

```
/*
 * NV_CTRL_GVI_MAX_CHANNELS_PER_JACK  - Returns the maximum
 * supported number of (logical) channels within a single physical jack
of
 * a GVI device.  For most SDI video formats, there is only one channel
 * (channel 0).  But for 3G video formats (as specified in SMPTE 425),
 * as an example, there are two channels (channel 0 and channel 1) per
 * physical jack.
 */

#define NV_CTRL_GVI_MAX_CHANNELS_PER_JACK                       336 /*
R--I */

/*
 * NV_CTRL_GVI_MAX_STREAMS  - Returns the maximum number of streams
 * that can be configured on the GVI device.
 */

#define NV_CTRL_GVI_MAX_STREAMS                                 337 /*
R--I */

/*
 * NV_CTRL_GVI_NUM_CAPTURE_SURFACES - The GVI interface exposed through
 * NV-CONTROL and the GLX_NV_video_input extension uses internal
capture
 * surfaces when frames are read from the GVI device.  The
 * NV_CTRL_GVI_NUM_CAPTURE_SURFACES can be used to query and assign the
 * number of capture surfaces.  This attribute is applied when
 * glXBindVideoCaptureDeviceNV() is called by the application.
 *
 * A lower number of capture surfaces will mean less video memory is
used,
 * but can result in frames being dropped if the application cannot
keep up
 * with the capture device.  A higher number will prevent frames from
being
 * dropped, making capture more reliable but will consume move video
memory.
 */
#define NV_CTRL_GVI_NUM_CAPTURE_SURFACES                        338 /*
RW-I */

/**************************************************************************
****/

/*
 * String Attributes:
 *
 * String attributes can be queryied through the
XNVCTRLQueryStringAttribute()
 * and XNVCTRLQueryTargetStringAttribute() function calls.
 *
 * String attributes can be set through the XNVCTRLSetStringAttribute()
 * function call.  (There are currently no string attributes that can
be
 * set on non-X Screen targets.)
 *
 * Unless otherwise noted, all string attributes can be queried/set
using an
 * NV_CTRL_TARGET_TYPE_X_SCREEN target.  Attributes that cannot take an
 * NV_CTRL_TARGET_TYPE_X_SCREEN target also cannot be queried/set
through
 * XNVCTRLQueryStringAttribute()/XNVCTRLSetStringAttribute() (Since
 * these assume an X Screen target).
 */
/*
 * NV_CTRL_STRING_GVIO_FIRMWARE_VERSION - indicates the version of the
 * Firmware on the GVIO device.
 */

#define NV_CTRL_STRING_GVIO_FIRMWARE_VERSION                    8  /*
R--I */

/*
 * The following is deprecated; use
NV_CTRL_STRING_GVIO_FIRMWARE_VERSION,
 * instead
 */
#define NV_CTRL_STRING_GVO_FIRMWARE_VERSION                     8  /*
R-- */
/*
 * NV_CTRL_STRING_GVIO_VIDEO_FORMAT_NAME - query the name for the
specified
 * NV_CTRL_GVIO_VIDEO_FORMAT_*.  So that this can be queried with
existing
 * interfaces, XNVCTRLQueryStringAttribute() should be used, and the
video
 * format specified in the display_mask field; eg:
 *
 * XNVCTRLQueryStringAttribute(dpy,
 *                             screen,
 *
NV_CTRL_GVIO_VIDEO_FORMAT_720P_60_00_SMPTE296,
 *                             NV_CTRL_GVIO_VIDEO_FORMAT_NAME,
 *                             &name);
 */

#define NV_CTRL_STRING_GVIO_VIDEO_FORMAT_NAME                  33  /*
R--GI */

/*
 * The following is deprecated; use
NV_CTRL_STRING_GVIO_VIDEO_FORMAT_NAME,
 * instead
 */
```

```
#define NV_CTRL_STRING_GVO_VIDEO_FORMAT_NAME                    33  /*
R--- */

#define NV_CTRL_STRING_LAST_ATTRIBUTE \
        NV_CTRL_STRING_GVIO_VIDEO_FORMAT_NAME

/**********************************************************************
****/

/*
 * String Operation Attributes:
 *
 * These attributes are used with the XNVCTRLStringOperation()
 * function; a string is specified as input, and a string is returned
 * as output.
 *
 * Unless otherwise noted, all attributes can be operated upon using
 * an NV_CTRL_TARGET_TYPE_X_SCREEN target.
 */

/*
 * NV_CTRL_STRING_OPERATION_GVI_CONFIGURE_STREAMS - Configure the
streams-
 * to-jack+channel topology for a GVI (Graphics capture board).
 *
 * The string input to GVI_CONFIGURE_STREAMS may be NULL.  If this is
the
 * case, then the current topology is returned.
 *
 * If the input string to GVI_CONFIGURE_STREAMS is not NULL, the string
 * is interpreted as a semicolon (";") separated list of comma-
separated
 * lists of "option=value" pairs that define a stream's composition.
The
 * available options and their values are:
 *
 *   "stream": Defines which stream this comma-separated list
describes.
 *             Valid values are the integers between 0 and
 *             NV_CTRL_GVI_NUM_STREAMS-1 (inclusive).
 *
 *   "linkN":  Defines a jack+channel pair to use for the given link N.
 *             Valid options are the string "linkN", where N is an
integer
 *             between 0 and NV_CTRL_GVI_MAX_LINKS_PER_STREAM-1
(inclusive).
 *             Valid values for these options are strings of the form
 *             "jackX" and/or "jackX.Y", where X is an integer between
0 and
 *             NV_CTRL_GVI_NUM_JACKS-1 (inclusive), and Y (optional) is
an
 *             integer between 0 and NV_CTRL_GVI_MAX_CHANNELS_PER_JACK-
1
```

```
 *              (inclusive).
 *
 * An example input string might look like:
 *
 *   "stream=0, link0=jack0, link1=jack1; stream=1, link0=jack2.1"
 *
 *   This example specifies two streams, stream 0 and stream 1.  Stream
0
 *   is defined to capture link0 data from the first channel (channel
0) of
 *   BNC jack 0 and link1 data from the first channel of BNC jack 1.
The
 *   second stream (Stream 1) is defined to capture link0 data from
channel 1
 *   (second channel) of BNC jack 2.
 *
 * This example shows a possible configuration for capturing 3G input:
 *
 *   "stream=0, link0=jack0.0, link1=jack0.1"
 *
 * Applications should query the following attributes to determine
 * possible combinations:
 *
 *   NV_CTRL_GVI_MAX_STREAMS
 *   NV_CTRL_GVI_MAX_LINKS_PER_STREAM
 *   NV_CTRL_GVI_NUM_JACKS
 *   NV_CTRL_GVI_MAX_CHANNELS_PER_JACK
 *
 * Note: A jack+channel pair can only be tied to one link/stream.
 *
 * Upon successful configuration or querying of this attribute, a
string
 * representing the current topology for all known streams on the
device
 * will be returned.  On failure, NULL is returned.
 */

#define NV_CTRL_STRING_OPERATION_GVI_CONFIGURE_STREAMS         4 /* RW-
I */
/**********************************************************************
****/

/*
 * CTRLAttributeValidValuesRec -
 *
 * structure and related defines used by
 * XNVCTRLQueryValidAttributeValues() to describe the valid values of
 * a particular attribute.  The type field will be one of:
 *
 * ATTRIBUTE_TYPE_INTEGER : the attribute is an integer value; there
 * is no fixed range of valid values.
 *
 * ATTRIBUTE_TYPE_BITMASK : the attribute is an integer value,
 * interpretted as a bitmask.
 *
 * ATTRIBUTE_TYPE_BOOL : the attribute is a boolean, valid values are
 * either 1 (on/true) or 0 (off/false).
 *
 * ATTRIBUTE_TYPE_RANGE : the attribute can have any integer value
 * between NVCTRLAttributeValidValues.u.range.min and
 * NVCTRLAttributeValidValues.u.range.max (inclusive).
 *
 * ATTRIBUTE_TYPE_INT_BITS : the attribute can only have certain
 * integer values, indicated by which bits in
 * NVCTRLAttributeValidValues.u.bits.ints are on (for example: if bit
 * 0 is on, then 0 is a valid value; if bit 5 is on, then 5 is a valid
 * value, etc).  This is useful for attributes like NV_CTRL_FSAA_MODE,
 * which can only have certain values, depending on GPU.
 *
 *
 * The permissions field of NVCTRLAttributeValidValuesRec is a bitmask
 * that may contain:
 *
 * ATTRIBUTE_TYPE_READ      - Attribute may be read (queried.)
 * ATTRIBUTE_TYPE_WRITE     - Attribute may be written to (set.)
 * ATTRIBUTE_TYPE_DISPLAY   - Attribute requires a display mask.
 * ATTRIBUTE_TYPE_GPU       - Attribute is valid for GPU target types.
 * ATTRIBUTE_TYPE_FRAMELOCK - Attribute is valid for Frame Lock target
types.
 * ATTRIBUTE_TYPE_X_SCREEN  - Attribute is valid for X Screen target
types.
 * ATTRIBUTE_TYPE_XINERAMA  - Attribute will be made consistent for all
 *                            X Screens when the Xinerama extension is
enabled.
 * ATTRIBUTE_TYPE_VCSC      - Attribute is valid for Visual Computing
System
 *                            target types.
 * ATTRIBUTE_TYPE_GVI       - Attribute is valid for Graphics Video In
target
 *                            types.
 *
 *
 * See 'Key to Integer Attribute "Permissions"' at the top of this
 * file for a description of what these permission bits mean.
 */

#define ATTRIBUTE_TYPE_UNKNOWN   0
#define ATTRIBUTE_TYPE_INTEGER   1
#define ATTRIBUTE_TYPE_BITMASK   2
#define ATTRIBUTE_TYPE_BOOL      3
#define ATTRIBUTE_TYPE_RANGE     4
#define ATTRIBUTE_TYPE_INT_BITS  5

#define ATTRIBUTE_TYPE_READ       0x001
#define ATTRIBUTE_TYPE_WRITE      0x002
#define ATTRIBUTE_TYPE_DISPLAY    0x004
```

```
#define ATTRIBUTE_TYPE_GPU        0x008
#define ATTRIBUTE_TYPE_FRAMELOCK  0x010
#define ATTRIBUTE_TYPE_X_SCREEN   0x020
#define ATTRIBUTE_TYPE_XINERAMA   0x040
#define ATTRIBUTE_TYPE_VCSC       0x080
#define ATTRIBUTE_TYPE_GVI        0x100

typedef struct _NVCTRLAttributeValidValues {
    int type;
    union {
        struct {
            int min;
            int max;
        } range;
        struct {
            unsigned int ints;
        } bits;
    } u;
    unsigned int permissions;
} NVCTRLAttributeValidValuesRec;


/**********************************************************************
****/

/*
 * NV-CONTROL X event notification.
 *
 * To receive X event notifications dealing with NV-CONTROL, you should
 * call XNVCtrlSelectNotify() with one of the following set as the type
 * of event to receive (see NVCtrlLib.h for more information):
 */

#define ATTRIBUTE_CHANGED_EVENT                     0
#define TARGET_ATTRIBUTE_CHANGED_EVENT              1
#define TARGET_ATTRIBUTE_AVAILABILITY_CHANGED_EVENT 2
#define TARGET_STRING_ATTRIBUTE_CHANGED_EVENT       3
#define TARGET_BINARY_ATTRIBUTE_CHANGED_EVENT       4
```

13 ANCILLARY DATA API

```
///////////////////////////////////////////////////////////////////////
//////
// ANCAPI.H
//
// Header file for ANCAPI.CPP - This header file implements the NVIDIA
GVO
// ancillary data API for SDI.
//
//  This file will be exposed to 3rd party developers
//
// Platforms/OS - Windows XP, linux
//
///////////////////////////////////////////////////////////////////////
//////

#ifndef __NVANCAPI_H__
#define __NVANCAPI_H__

#ifdef _WIN32
#include "nvapi.h"
#endif

//---------------------------------------------------------------------
// NVIDIA Grapics to Video Out (GVO) Ancillary Data API
//---------------------------------------------------------------------

#ifdef  __cplusplus
    extern "C" {
#endif//__cplusplus

#ifndef IN
#   define IN
#endif//IN

#ifndef OUT
```

```
#   define OUT
#endif//OUT

#ifndef INOUT
#   define INOUT
#endif//INOUT

#ifdef _WIN32
#define NVVIOANCAPI_INTERFACE extern NvAPI_Status __cdecl
#else
#define NVVIOANCAPI_INTERFACE NvAPI_Status
#endif

#define DECLARE_HANDLE(name) struct name##__ { int unused; }; typedef
struct name##__ *name


// Need these nvapi.h defines on linux.
#ifndef _WIN32
typedef unsigned long long NvU64;
typedef unsigned int       NvU32;
typedef unsigned short     NvU16;
typedef long               NvS32;
typedef unsigned char      NvU8;

#define NVAPI_GENERIC_STRING_MAX    4096
#define NVAPI_LONG_STRING_MAX       256
#define NVAPI_SHORT_STRING_MAX      64

typedef char NvAPI_String[NVAPI_GENERIC_STRING_MAX];
typedef char NvAPI_LongString[NVAPI_LONG_STRING_MAX];
typedef char NvAPI_ShortString[NVAPI_SHORT_STRING_MAX];
#endif


//
// NVVIO Handle - NVVIO control handle
//
#ifndef _WIN32
DECLARE_HANDLE(NvVioHandle);            // NvVIO Device Handle
#endif

//
=======================================================================
==================
// NvAPI Version Definition
// Maintain per structure specific version define using the
MAKE_NVAPI_VERSION macro.
// Usage: #define NVVIOANCDATAFRAME_VERSION
MAKE_NVAPI_VERSION(NVVIOANCDATAFRAME, 1)
//
=======================================================================
==================
#define MAKE_NVAPI_VERSION(typeName,ver) (NvU32)(sizeof(typeName) |
((ver)<<16))
#define GET_NVAPI_VERSION(ver) (NvU32)((ver)>>16)
#define GET_NVAPI_SIZE(ver) (NvU32)((ver) & 0xffff)

//----------------------------------------------------------------------
// Types
//----------------------------------------------------------------------

//----------------------------------------------------------------------
// Enumerations
//----------------------------------------------------------------------

// ====================================================
// NvAPI Status Values
//    All NvAPI functions return one of these codes.
// ====================================================
#ifndef _WIN32
#ifndef NvAPI_Status
typedef enum
{
    NVAPI_OK                            =  0,      // Success
    NVAPI_ERROR                         = -1,      // Generic error
    NVAPI_LIBRARY_NOT_FOUND             = -2,      // nvapi.dll can not
be loaded
    NVAPI_NO_IMPLEMENTATION             = -3,      // not implemented
in current driver installation
    NVAPI_API_NOT_INTIALIZED            = -4,      // NvAPI_Initialize
has not been called (successfully)
    NVAPI_INVALID_ARGUMENT              = -5,      // invalid argument
    NVAPI_NVIDIA_DEVICE_NOT_FOUND       = -6,      // no NVIDIA display
driver was found
    NVAPI_END_ENUMERATION               = -7,      // no more to enum
    NVAPI_INVALID_HANDLE                = -8,      // invalid handle
    NVAPI_INCOMPATIBLE_STRUCT_VERSION   = -9,      // an argument's
structure version is not supported
    NVAPI_NOT_SUPPORTED                 = -10,     // Requested feature
not supported in the selected GPU
    NVAPI_PORTID_NOT_FOUND              = -11      // NO port ID found
for I2C transaction
} NvAPI_Status;
#endif
#endif

// Audio sample rate definitions - from SMPTE 299M-2004 Table 8
typedef enum
{
      NVVIOANCAUDIO_SAMPLING_RATE_48_0         = 0x0,
      NVVIOANCAUDIO_SAMPLING_RATE_44_1         = 0x1,
      NVGOVANCAUDIO_SAMPLING_RATE_32_0         = 0x2,
      NVVIOANCAUDIO_SAMPLING_RATE_FREE_RUNNING = 0x3
} NVVIOANCAUDIO_SAMPLE_RATE;
```

```
// Active channel definitions - from SMPTE 299M-2004 Table 9
typedef enum
{
    NVVIOANCAUDIO_ACTIVE_CH1    = 0x1,
    NVVIOANCAUDIO_ACTIVE_CH2    = 0x2,
    NVVIOANCAUDIO_ACTIVE_CH3    = 0x4,
    NVVIOANCAUDIO_ACTIVE_CH4    = 0x8
} NVVIOANCAUDIO_ACTIVE_CHANNEL;


//----------------------------------------------------------------------
// Structures
//----------------------------------------------------------------------

// Audio control
typedef struct tagNVVIOANCAUDIOCNTRL {
    NvU32 version;                // Structure version
    NvU8 frameNumber1_2;          // Frame number for channels 1 and 2
    NvU8 frameNumber3_4;          // Frame number for channels 3 and 4
    NvU8 rate;                    // Audio sample rate
    NvU8 asynchronous;            // 0 = synchronous, 1 = asynchronous
    NvU8 activeChannels;          // Bitwise OR of active channel
definitions
} NVVIOANCAUDIOCNTRL;

#define NVVIOANCAUDIOCNTRL_VERSION
MAKE_NVAPI_VERSION(NVVIOANCAUDIOCNTRL, 1)

// Audio group
typedef struct tagNVVIOANCAUDIOGROUP {
    NvU32 numAudioSamples;    // Number of valid audio samples / channel
    NvU32  *audioData[4];     // Data pointer for audio channels 1-4
    NVVIOANCAUDIOCNTRL audioCntrl;  // Controls for audio channels 1-4
} NVVIOANCAUDIOGROUP;

#define NVVIOANCAUDIOGROUP_VERSION
MAKE_NVAPI_VERSION(NVVIOANCAUDIOGROUP, 1)

// Per ANC Data Packet
typedef struct tagNVVIOANCDATAPACKET {
    NvU32 version;                // Structure version
    NvU16 DID;
    NvU16 SDID;
    NvU16 DC;
    NvU8  *data;                  // Should this be unsigned short?
    NvU16 CS;
} NVVIOANCDATAPACKET;

#define NVVIOANCDATAPACKET_VERSION
MAKE_NVAPI_VERSION(NVVIOANCDATAPACKET, 1)

// Data field mask definitions (Indicate NVVIOANCDATAFRAME fields in
use)
```

```
#define NVVIOANCDATAFRAME_AUDIO_GROUP_1     0x00000001
#define NVVIOANCDATAFRAME_AUDIO_GROUP_2     0x00000002
#define NVVIOANCDATAFRAME_AUDIO_GROUP_3     0x00000004
#define NVVIOANCDATAFRAME_AUDIO_GROUP_4     0x00000008
#define NVVIOANCDATAFRAME_LTC           0x00000010
#define NVVIOANCDATAFRAME_VITC          0x00000020
#define NVVIOANCDATAFRAME_FILM_TC       0x00000040
#define NVVIOANCDATAFRAME_PROD_TC       0x00000080
#define NVVIOANCDATAFRAME_FRAME_ID      0x00000100
#define NVVIOANCDATAFRAME_CUSTOM        0x00000200

typedef struct tagNVVIOANCDATAFRAME {
    NvU32 version;                    // Structure version
    NvU32 fields;                    // Field mask
    NVVIOANCAUDIOGROUP  AudioGroup1; // Audio group 1
    NVVIOANCAUDIOGROUP  AudioGroup2; // Audio group 2
    NVVIOANCAUDIOGROUP  AudioGroup3; // Audio group 3
    NVVIOANCAUDIOGROUP  AudioGroup4; // Audio group 4
    NvU32 LTCTimecode;               // RP188
    NvU32 LTCUserBytes;
    NvU32 VITCTimecode;
    NvU32 VITCUserBytes;
    NvU32 FilmTimecode;
    NvU32 FilmUserBytes;
    NvU32 ProductionTimecode;        // RP201
    NvU32 ProductionUserBytes;       // RP201
    NvU32 FrameID;
    NvU32 numCustomPackets;
    NVVIOANCDATAPACKET *CustomPackets;
} NVVIOANCDATAFRAME;

#define NVVIOANCDATAFRAME_VERSION
MAKE_NVAPI_VERSION(NVVIOANCDATAFRAME, 1)

// Per Sequence
typedef struct tagNVVIOANCDATACONFIG {
    NvU32 version;               // Structure version
    NvU32 numAudioChannels;
    NvU32 audioRate;
} NVVIOANCDATACONFIG;

#define NVVIOANCDATACONFIG_VERSION
MAKE_NVAPI_VERSION(NVVIOANCDATACONFIG, 1)

//--------------------------------------------------------------------
// Prototypes
//--------------------------------------------------------------------

//////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_InitializeGVO
```

```
//
//   DESCRIPTION: Initializes NV GVO ancillary data library. This
function must be
//                called before any other NV GVO ancillary data library
function.
//                This function queries the current video device state
and
//                initializes all internal data structures.
//
//
// RETURN STATUS: NVAPI_ERROR            Something is wrong during the
initialization process (generic error)
//                NVAPI_LIBRARYNOTFOUND  Can not load nvapi.dll
//                NVAPI_OK               Initialized
//
////////////////////////////////////////////////////////////////////////
////////
#ifdef _WIN32
NVVIOANCAPI_INTERFACE NvVIOANCAPI_InitializeGVO(NvVioHandle hVIO)
#else
NVVIOANCAPI_INTERFACE NvVIOANCAPI_InitializeGVO(Display *dpy, int
target_id)
#endif

////////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_InitializeGVI
//
//   DESCRIPTION: Initializes NV GVI ancillary data library. This
function must be
//                called before any other NV GVI ancillary data library
function.
//                This function queries the current video device state
and
//                initializes all internal data structures.
//
//
// RETURN STATUS: NVAPI_ERROR            Something is wrong during the
initialization process (generic error)
//                NVAPI_LIBRARYNOTFOUND  Can not load nvapi.dll
//                NVAPI_OK               Initialized
//
////////////////////////////////////////////////////////////////////////
////////
#ifdef _WIN32
NVVIOANCAPI_INTERFACE NvVIOANCAPI_InitializeGVI(NvVioHandle hVIO)
#else
NVVIOANCAPI_INTERFACE NvVIOANCAPI_InitializeGVI(Display *dpy, int
target_id)
#endif
```

```
///////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_ReleaseGVO
//
//   DESCRIPTION: Releases NV GVO ancillary data library. This function
must be
//                called to release all NV GVO ancillary data library
resources.
//
//
// RETURN STATUS: NVAPI_ERROR            Something went wrong
//                NVAPI_OK               All resources released
//
///////////////////////////////////////////////////////////////////////
////////
#ifdef _WIN32
NVVIOANCAPI_INTERFACE NvVIOANCAPI_ReleaseGVO(NvVioHandle hVIO)
#else
NVVIOANCAPI_INTERFACE NvVIOANCAPI_ReleaseGVO(Display *dpy, int
target_id)
#endif

///////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_ReleaseGVI
//
//   DESCRIPTION: Releases NV GVI ancillary data library. This function
must be
//                called to release all NV GVO ancillary data library
resources.
//
//
// RETURN STATUS: NVAPI_ERROR            Something went wrong
//                NVAPI_OK               All resources released
//
///////////////////////////////////////////////////////////////////////
////////
#ifdef WIN32
NVVIOANCAPI_INTERFACE NvVIOANCAPI_ReleaseGVI(NvVioHandle hVIO)
#else
NVVIOANCAPI_INTERFACE NvVIOANCAPI_ReleaseGVI(Display *dpy, int
target_id)
#endif


///////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_GetErrorMessage
//
```

```
//   DESCRIPTION: converts an NVVIOANCAPI error code into a null
terminated string
//
// RETURN STATUS: null terminated string (always, never NULL)
//
///////////////////////////////////////////////////////////////////////
////////
NVVIOANCAPI_INTERFACE NvVIOANCAPI_GetErrorMessage(NvAPI_Status
nr,NvAPI_ShortString szDesc);

///////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_GetInterfaceVersionString
//
//   DESCRIPTION: Returns a string describing the version of the
NVVIOANCAPI library.
//                Contents of the string are human readable.  Do not
assume a fixed
//                format.
//
// RETURN STATUS: User readable string giving info on NvAPI's version
//
///////////////////////////////////////////////////////////////////////
////////
NVVIOANCAPI_INTERFACE
NvVIOANCAPI_GetInterfaceVersionString(NvAPI_ShortString szVersion);


///////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_SendANCData
//
//   DESCRIPTION: Sends ancillary data for current field or frame.
//
// RETURN STATUS: NVAPI_ERROR
//                NVAPI_OK
//
///////////////////////////////////////////////////////////////////////
////////
NVVIOANCAPI_INTERFACE NvVIOANCAPI_SendANCData(NvVioHandle handle,
NVVIOANCDATAFRAME *data);


///////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_NumAudioSamples
//
//   DESCRIPTION: Return number of expected audio samples per channel
per frame
//                at the given sample rate.
```

```
//
// RETURN STATUS: NVAPI_ERROR
//                NVAPI_OK
//
//////////////////////////////////////////////////////////////////////////
////////
NVVIOANCAPI_INTERFACE NvVIOANCAPI_NumAudioSamples(NvVioHandle handle,
NVVIOANCAUDIO_SAMPLE_RATE rate, float *num);

//////////////////////////////////////////////////////////////////////////
////////
//
// FUNCTION NAME: NvVIOANCAPI_CaptureANCData
//
//   DESCRIPTION: Capture ancillary data for current field or frame.
//
// RETURN STATUS: NVAPI_ERROR
//                NVAPI_OK
//
//////////////////////////////////////////////////////////////////////////
////////
NVVIOANCAPI_INTERFACE NvVIOANCAPI_CaptureANCData(NvVioHandle handle,
NVVIOANCDATAFRAME *data);

#ifdef __cplusplus
}; //extern "C" {
#endif

#endif  // __NVANCAPI_H__
```

Notice

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

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

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

ROVI Compliance Statement

NVIDIA Products that are ROVI-enabled can only be sold or distributed to buyers with a valid and existing authorization from ROVI to purchase and incorporate the device into buyer’s products.

This device is protected by U.S. patent numbers 6,516,132; 5,583,936; 6,836,549; 7,050,698; and 7,492,896 and other intellectual property rights. The use of ROVI Corporation's copy protection technology in the device must be authorized by ROVI Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by ROVI Corporation. Reverse engineering or disassembly is prohibited.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

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

Copyright