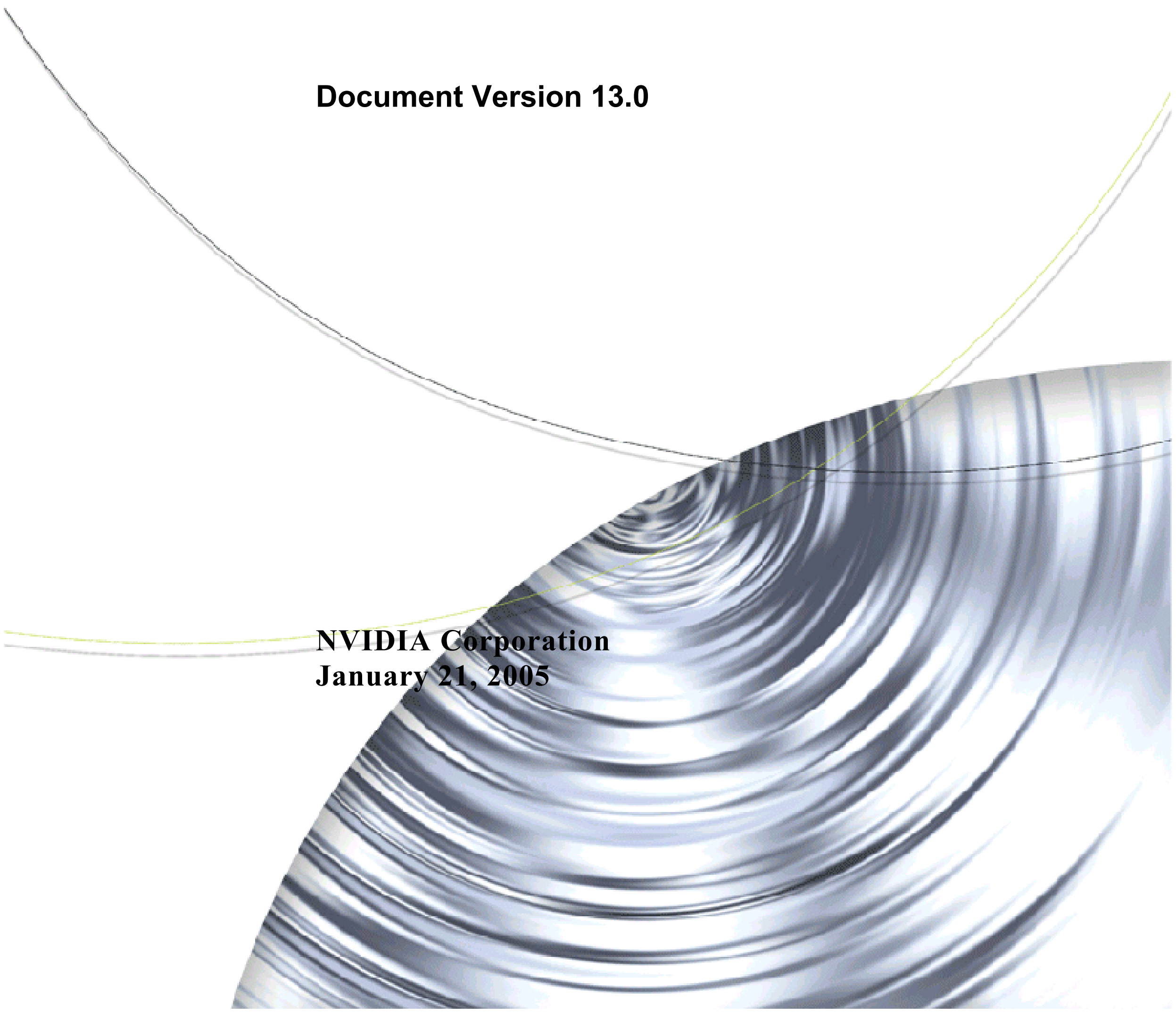
UTILITIES AND APIS

NVCPL.DLL API Manual



Document Version 13.0

NVIDIA Corporation
January 21, 2005

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

Table of Contents

1 OVERVIEW

About This Document

This document describes several APIs and functions that are exposed by the NVIDIA driver component `nvcpl.dll`. The document contains the following sections:

- Command Line Functions

  This chapter describes the command line functions that allows configuration of the desktop and its displays using the Windows Start->Run dialog box.

- Control Panel APIs

  This chapter describes several APIs that allow you to control the display gamma, the display PowerMizer settings, and also to obtain display information such as multimonitor modes and a list of the displays that are connected to the system.

- IDispatch COM Interface

  This chapter describes the COM IDispatch interface for controlling several TV-out settings.

Document Revision History

Revision	Date	Description
1.0	1/28/03	Initial Release. Combined previous PowerMizer API with DTCFG document. Added new APIs.
2.0	2/7/03	Added `launchtvapplet`, `setvof`, and `NvCplGetFlatPanelNativeRes().`
3.0	6/24/03	Added `setappprofiles, setvsync, queryappprofiles` command line functions. Updated PowerMizer and Gamma Ramp APIs.
4.0	10/13/03	Added the following command line functions: `launchpowemizerapplet, nvcplForceTVConnection, NvCycleDisplayDevice, NvCycleDisplayDeviceEx, ExportNvWsApps.` Added the following **dtcfg** functions: `getdvcvalue, getbrightnessvalue, getcontrastvalue, getgammavalue, geticm, seticm, hires.` Added the following APIs: `dtcfgex(), GetdtcfgLastError(), GetdtcfgLastErrorEx(), NvGetFullScreenVideoMirrorEnabled(), NvSetFullScreenVideoMirrorEnabled(), NvCplGetScalingStatus(), NvCplSetDataInt(), NvCplGetDataInt(), NvCplGetThermalSettings()`
4.1	10/26/03	Fixed API names: `NvGetFullScreenVideoMirroringEnabled(), NvSetFullScreenVideoMirroringEnabled(),`
5.0	11/14/03	Added NvcplLatebound chapter. Other corrections.
5.1	11/19/03	Clarified the description of NvcplLateBound interface.
6.0	12/03/03	Added `NvGetCurrentTVFormat()`; Added `dtcfg settvformat`.
6.1	12/23/03	Further clean up of NvcplLatebound chapter. Removed unsupported examples in the code samples.
6.2	2/13/04	NvcplLatebound changes - fixed order of TV positioning arguments.
(7.0) R55-1.0	1/9/04	NVCPL.DLL Manual for Release 55. Added: • `SelectDisplayDevice()` calls to Control Panel APIs and IDispatch COM Interface chapters. • `NvGetDisplayInfo()` • `SetHDAspect()`
(8.0) R55-2.0	2/12/04	Added "NvSetTVWideScreenSignalling()" on page 70. Added "NvTVContentProtection()" on page 69. (Removed in version 13.0)

Revision	Date	Description
(8.1) R55-2.1	2/27/04	Revised "NvSetTVWideScreenSignalling()" on page 70, "NvTVContentProtection()" on page 69, and "IDispatch COM Interface" on page 83.
(9.0 preliminary) R60-1.0	3/03/04	NVCPL.DLL Manual for Release 60. Added: • `setsharpness` and "set_normalize_sharpness" on page 21 • `getsharpnessvalue` and "get_normalize_sharpnessvalue" on page 28 • "NvColorGetGammaRampEx()" on page 47 and "NvColorSetGammaRampEx()" on page 48
9.0	5/07/04	Consolidated Release 55/60 versions into one document. Added to API calls: • "getdvihdformat" on page 25 • "NvGetTVConnectedStatus()" on page 68 • "NVTVOutManageOverscanConfiguration()" on page 71 Added to NvCplLatebound supported calls: • "Saturation" on page 86 to TV controls • "Calls Corresponding to Panel APIs" on page 84 Updated: • "settvformat" on page 18
10.0	6/04/04	Added • "NvCplIsExternalPowerConnectorAttached()" on page 65 • "setscreenposition" on page 17
11.0	8/12/04	Added: • "setscreenposition" on page 17 • "NvCplGetMSOrdinalDeviceString()" on page 37 Updated: • "NvGetDisplayInfo()" on page 30 • "NvCplSetDataInt()" on page 67

Revision	Date	Description
12.0	10/3/04	Added: • "NvEnumDisplaySettings()" on page 38 • "NvGetDisplayCustomName()" on page 40 • "NvSetDisplayCustomName()" on page 41 • "NvGetLastDisplaySettings()" on page 42 • "NvGetDefaultDisplaySettings()" on page 43 • "NvCplGetRealConnectedDevicesString()" on page 56 • "NvGetPhysicalConnectorInfo()" on page 59 • "NvEnumPhysicalConnectorDetails()" on page 60 • "NvCplRefreshConnectedDevices()" on page 55 • "NvCplGetActiveDevicesString()" on page 58 • "Result Codes" on page 79 • "Code Samples" on page 104 • "NVCPL API Device Moniker Specification Version 2" on page 97 Updated: • "NvGetDisplayInfo()" on page 30
13.0	1/21/05	Updated • "setscreenposition" on page 17 • "Data Control" on page 66 Added "TV VBI Functions" on page 72

System Requirements

This document applies to `NVCPL.DLL` APIs for NVIDIA Forceware graphics drivers for Windows® 95, Windows 98, Windows Me, Windows NT 4.0, Windows 2000 and Windows XP.

2 Command Line Functions

Overview of Exported Functions

The NVIDIA Control Panel library exports command line functions that allows configuration of the desktop and its displays using the Windows Start->Run dialog box. NVIDIA control panel interfaces are exposed to this API as individual commands.

The command line interface uses the following default Windows callback prototype:

```
void APIENTRY EntryPoint(
     HWND hwnd,          // handle to owner window
     HINSTANCE hinst,    // instance handle for the DLL
     LPTSTR lpszCmdLine, // string that the DLL parses
     int nCmdShow        // show state
     );
```

Currently, the following functions and controls are supported:

- Desktop Manipulation
- PowerMizer Page
- TV Settings
- Workstation Application Profile
- Vertical Sync Control

Desktop Manipulation

dtcfg

DTCFG was developed to assist in manual testing and verification of desktop display behavior such as nView display modes, rotation, and digital vibrance settings.

DTCFG is explained in the following sections:

- "Using DTCFG—Setting Delay Times" on page 8 explains how to coordinate multiple commands.
- "Using DTCFG—Configuring the Desktop" on page 9 describes each of the DTCFG commands for configuring the desktop.

Description	Configure the desktop.
Format	`rundll32.exe NvCpl.dll,dtcfg <command> <display#> [arg1] [arg2] [arg3] [arg4]`
Comments	See "Using DTCFG—Configuring the Desktop" on page 9 for details.

NvCycleDisplayDevice

Description	Cycle the displays between the CRT and the TV.
Format	`rundll32.exe NvCpl.dll,NvCycleDisplayDevice [display#] [head#]`
Comments	*display#* is the display number on the Windows Settings page. This can be any of the values shown on the Windows Settings page, or a value of 0 for the current Windows primary display. *head#* applies only to nView Clone or Spanning mode, and specifies the nView display (1 or 2).

NvCycleDisplayDeviceEx

Description	Cycle the displays between the CRT, the TV, and the DFP.
Format	`rundll32.exe NvCpl.dll,NvCycleDisplayDeviceEx [display#] [head#]`
Comments	*display#* is the display number on the Windows Settings page. This can be any of the values shown on the Windows Settings page, or a value of 0 for the current Windows primary display. *head#* applies only to nView Clone or Spanning mode, and specifies the nView display (1 or 2).

PowerMizer Page

`launchpowermizerapplet`

Description Launches the NVIDIA Display Properties PowerMizer page.

Format `rundll32.exe NvCpl.dll,launchpowermizerapplet`

TV Settings

`launchtvapplet`

Description Open the NVIDIA Display Properties TV Settings page.

Format `rundll32.exe NvCpl.dll,launchtvapplet`

`NvCplForceTVConnection`

Description Forces a TV connection.

Format `rundll32.exe NvCpl.dll,NvCplForceTVConnection`

Workstation Application Profile

`ExportNvWsApps`

Description Exports the selected workstation application settings to the file NvWsApps.txt in the Windows System32 directory.

Format `rundll32.exe NvCpl.dll,ExportNvWsApps`

`setappprofile`

Description Set the workstation application profile.

Format `rundll32.exe NvCpl.dll,setappprofile <`*`profile`*`>`

Comments *`profile`* must be one of the available workstation profiles, or “default” to reset to the default profile.

`queryappprofiles`

Description Query the available application profiles—those listed in the configuration file `nvwsapps.cfg`—and store the names in the specified text file.

Format	`rundll32.exe NvCpl.dll,queryappprofiles` *`c:\\ profiles.txt`*
Comments	'*`c:\\profiles.txt`*' is the path and filename where the profile names are to be stored.

Vertical Sync Control

setvsync

Description	Turn V-Sync ON or OFF.
Format	`rundll32.exe NvCpl.dll,setvsync on|off`

Using DTCFG—Setting Delay Times

You can run several DTCFG commands from a batch file. To make sure that the commands are launched in a controlled manner, and to avoid conflicts between commands, you can impose a delay time between commands.

Even though multiple commands will always run serially, specifying a delay is useful when you want to ensure that a process—such as a modeset—has enough time to complete.

Command Description

To configure the delay time between **`dtcfg`** commands, enter the command line in the following format:

"`rundll32.exe NvCpl.dll,dtcfg setdelay` *`<delay_type>`* *`<delay_time>`*"

where

- *`delay_type`* is one of the following:

  `pre` - indicates the delay time is imposed *before* each command is processed.

  `post` - indicates the delay time is imposed *after* each command is processed.

- *`delay_time`* is the time in milliseconds.

Note: Use of the **`setdelay`** command can exaggerate the serial execution of multiple commands.

The delay time specified by the **`setdelay`** command applies to all subsequent commands, and can be changed only by reissuing the `setdelay` command using a different time value. To restore the wait time to zero, issue the **`setdelay`** command using a *`delay_time`* value of 0.

Examples

- **`rundll32.exe NvCpl.dll,dtcfg setdelay pre 500`**

  Wait 0.5 seconds before starting the next process.

- **`rundll32.exe NvCpl.dll,dtcfg setdelay post 120000`**

  Wait 2 minutes after a process completes before starting the next process.

Using DTCFG—Configuring the Desktop

To configure different displays and the desktop, enter the command line in the following format:

“**`rundll32.exe NvCpl.dll,dtcfg`** ***`<command>`*** ***`<display#>`*** **`[arg1] [arg2] [arg3] [arg4]`**”

where

- *`<command>`* can be any of the commands listed in the section “DTCFG Desktop Configuration Commands” on page 10.
- *`<display#>`* is the display number on the Windows Settings page.

  This can be any of the values shown on the Windows Settings page, a value of 0 for the current Windows primary display, or the word "all" for all of the displays on the Windows Settings page.

  Note: Most commands do not support the "all" option.

- `[`*`arg`***`n`**`]` varies depending on the command.

  See “Using NV Device Monikers” on page 9 for a description of this optional argument.

Using NV Device Monikers

Some commands use the optional `[`*`<NV device moniker>`*`]`. This indicates which nView display to apply the command. Under nView Clone or Spanning mode, if no mnemonic is specified then the command is applied to all the devices.

See “Device Moniker Version 2 String Format” on page 99 for a description of the format to use.

DTCFG Desktop Configuration Commands

Command Summary

Display Device Setup

- `attach` - Attach a display to the desktop
- `detach` - Detach a display from the desktop
- `detect` - Detect devices attached to the adapter backing a display
- `primary` - Make a display the current windows primary display
- `setview` - Set one of the TwinView modes on a display
- `setdefaults` - Sets the specified display parameter to the driver default value.

Display, TV, and Video Settings

- `rotate` - Rotate a display
- `setmode` - Set the display mode on a display
- `setscaling` - Sets the scaling of a display.
- `settvformat` - Sets the TV format standard of a display.
- `setscreenposition` - Sets the TV or CRT screen position.
- `svof` - Set the video output format.

Display Quality and Color Correction Settings

- `setdvc` - Set the Digital Vibrance level on a display
- `setgamma` - Set the gamma for the desired color channel.
- `setcontrast` - Set the contrast for the desired color channel.
- `set_normalize_contrast` - Set the contrast for the desired color channel, using normalized values.
- `setbrightness` - Set the brightness level for the desired color channel.
- `set_normalize_brightness` - Set the brightness level for the desired color channel, using normalized values.
- `setsharpness` - Set the sharpness value of the specified display.
- `set_normalize_sharpness` - Set the sharpness value of the specified display, using normalized values.
- `geticm` - Gets the ICC format of the display.
- `seticm` - Sets the display to the specified ICC format.

Following are detailed descriptions of each <*command*>.

attach

Description Attach a display to the desktop. This command disables nView multiview modes, such as Clone or Spanning, and enables Dualview.

Format `rundll32.exe NvCpl.dll,dtcfg attach <`*`display#`*`>`

Example
```
rundll32.exe NvCpl.dll,dtcfg attach 2
```
Attaches display #2 to the desktop. (Can also be accomplished using the Windows Settings page).

detach

Description Detach a display from the desktop.

Format `rundll32.exe NvCpl.dll,dtcfg detach <`*`display#`*`>`

Example
```
rundll32.exe NvCpl.dll,dtcfg detach 2
```
Detaches display #2 from the desktop. (Can also be accomplished using the Windows settings page).

detect

Description Scan for devices in order to detect which devices are attached to the adapter driving a display.

Format `rundll32.exe NvCpl.dll,dtcfg detect <`*`display#`*`>`

Example
```
rundll32.exe NvCpl.dll,dtcfg detect 2
```
Detects all displays attached to the adapter driving display #2. This command will normally not be used by most users.

geticm

Description Gets the ICC profile of the specified display. The filename is displayed on the command line. If no ICC profile is applied to the display, this command returns an empty string.

Format `rundll32.exe NvCpl.dll,dtcfg geticm <`*`NV device moniker`*`>`

Example
```
rundll32.exe NvCpl.dll,dtcfg geticm A0
```
Gets the ICC color format filename that is applied to CRT0.

hires

Description	Either prints the current high resolution scaling status to the commandline, or can be used to turn the high resolution scaling ON or OFF.
Format	`rundll32.exe NvCpl.dll,dtcfg hires [<ON|OFF>]`
Example 1	**`rundll32.exe NvCpl.dll,dtcfg hires`** Prints 'on' if the HiRes feature is turned on. Prints 'off' if the HiRes feature is turned off. Prints 'na' if the HiRes feature is not supported.
Example 2	**`rundll32.exe NvCpl.dll,dtcfg hires on`** Turns on the HiRes feature.
Example 3	**`rundll32.exe NvCpl.dll,dtcfg hires off`** Turns off the HiRes feature.

primary

Description	Make a display the current windows primary display.
Format	`rundll32.exe NvCpl.dll,dtcfg primary <display#>`
Example	**`rundll32.exe NvCpl.dll,dtcfg primary 2`** Makes display #2 the desktop primary. (Can also be accomplished using the windows settings page). If the display is not currently attached, it will be attached and then changed to the primary.

rotate

Description	Rotate a display to the designated orientation.
Format	`rundll32.exe NvCpl.dll,dtcfg rotate <display#> <angle: 0,90,180,270>`
Example	**`rundll32.exe NvCpl.dll,dtcfg rotate 2 90`** Rotates display #2 to the 90 degree position.

setbrightness

Description Set the brightness level of the specified display. This command overrides previous seticm settings.

Format

```
rundll32.exe NvCpl.dll,dtcfg setbrightness
<display#> [<NV device moniker>] <color channel>
<value:-125—125>
```

Where

- *color channel* is one of the following:
  - red
  - blue
  - green
  - all
- *value* is in the range -125 through 125.

Example

```
rundll32.exe NvCpl.dll,dtcfg setbrightness 2 all 100
```

Sets the brightness level for all color channels on display #2 to 100.

set_normalize_brightness

Description Set the brightness level of the specified display. This command is an alternative to `setbrightness` and uses normalized values. This command overrides previous seticm settings.

Format

```
rundll32.exe NvCpl.dll,dtcfg setbrightness
<display#> [<NV device moniker>] <color channel>
<value: 0.0—1.0>
```

Where

- *color channel* is one of the following:
  - red
  - blue
  - green
  - all
- *value* is in the range 0.0 through 1.0.

Example

```
rundll32.exe NvCpl.dll,dtcfg setbrightness 2 all 0.7
```

Sets the brightness level for all color channels on display #2 to 0.7.

setcontrast

Description Set the contrast level of the specified display. This command overrides previous seticm settings.

Format `rundll32.exe NvCpl.dll,dtcfg setcontrast` *`<display#>`* `[`*`<NV device moniker>`*`]` *`<color channel>`* *`<value: -82—82>`*

Where

- *`color channel`* is one of the following:
  - `red`
  - `blue`
  - `green`
  - `all`
- *`value`* is in the range -82 through 82.

Example **`rundll32.exe NvCpl.dll,dtcfg setcontrast 2 all 50`**

Sets the contrast level for all color channels on display #2 to 50.

set_normalize_contrast

Description Set the contrast level of the specified display. This command is an alternative to `setcontrast` and uses normalized values. This command overrides previous seticm settings.

Format `rundll32.exe NvCpl.dll,dtcfg setcontrast` *`<display#>`* `[`*`<NV device moniker>`*`]` *`<color channel>`* *`<value: 0.0—1.0>`*

Where

- *`color channel`* is one of the following:
  - `red`
  - `blue`
  - `green`
  - `all`
- *`value`* is in the range 0.0 through 1.0.

Example **`rundll32.exe NvCpl.dll,dtcfg setcontrast 2 all 0.6`**

Sets the contrast level for all color channels on display #2 to 0.6.

setdefaults

Earliest Driver 61.10

Description Restores the specified device and parameter to the driver default values.

Format rundll32.exe NvCpl.dll,dtcfg setdefaults *<display#>* [*<NV device moniker>*] *<option>*

Where

- *option* -- one of the following parameters currently supported in this call:
  - color - specifies all the parameters on the Color Correction page.
  - screenposition - specifies the screen position (applies only to CRT or TV)
  - timingmodesforcurrentmode - remove the advanced timings for the current resolution. To remove timings that were added with the settvformat command, the display must be in the corresponding mode. ***Effective as of 61.20***

Example **rundll32.exe NvCpl.dll,dtcfg setdefaults TA color**

Sets the TV color correction settings to the default values. .

setdvc

Description Set the Digital Vibrance level of the specified display.

Format rundll32.exe NvCpl.dll,dtcfg setdvc *<display#>* [*<NV device moniker>*] *<dvc value: 0-63>*

Where *<dvc value>* is a value from 0–60. If the adapter only supports DVC1 (value of 0–3), the value will be scaled as follows:

- 0–15: 0
- 16–31: 1
- 32–47: 2
- 48–63: 3

Example 1 **rundll32.exe NvCpl.dll,dtcfg setdvc 2 16**

Sets the digital vibrance on display #2 to 20. On an adaptor that supports only DVC1, the value of 16 is scaled to 1.

Example 2 **rundll32.exe NvCpl.dll,dtcfg setdvc all 16**

Sets the digital vibrance on all of the displays supporting DVC to 20. On an adaptor that supports only DVC1, the value of 16 is scaled to 1.

seticm

Description Applies an ICC profile to the specified display. This command overrides previous setbrightness/setcontrast/setgamma settings

Format `rundll32.exe NvCpl.dll,dtcfg seticm <NV device moniker> <ICC profile name>`

Where `<ICC profile name>` is the file that contains color correction information in industry standard ICC format. Common extensions for ICC profiles are `.ICC` and `.ICM`.

Example **`rundll32.exe NvCpl.dll,dtcfg seticm A0 ColorMatch.icc`**

Applies the ICC profile `ColorMatchRGB.icc` to CRT0.

setgamma

Description Set the gamma level of the specified display. This command overrides previous seticm settings.

Format `rundll32.exe NvCpl.dll,dtcfg setgamma <display#> [<NV device moniker>] <color channel> <value:0.5-6.0>`

Where

- *`color channel`* is one of the following:
  - `red`
  - `blue`
  - `green`
  - `all`
- *`value`* is in the range 0.5–6.0.

Example **`rundll32.exe NvCpl.dll,dtcfg setgamma 2 all 1.0`**

Sets the gamma value for all color channels on display #2 to 1.0.

setmode

Description Sets the display mode on a display.

Format `rundll32.exe NvCpl.dll,dtcfg setmode <display#> <hres> <vres> <bpp> <freq>`

Example **`rundll32.exe NvCpl.dll,dtcfg setmode 2 1024 768 32 75`**

Sets display #2 to 1024x768x32 @75Hz . (Can also be accomplished using the windows settings page).

setscaling

Description Set the scaling of the specified display.

Format `rundll32.exe NvCpl.dll,dtcfg setscaling <display#> [<NV device moniker>] <mode: 0,1, 2, 3, 5>`

The scaling modes are defined as follows:

- 0 : Default
- 1: Native
- 2: Scaled
- 3: Centered
- 5: Aspect scaling (for wide panel LCD)

Example **`rundll32.exe NvCpl.dll,dtcfg setscaling 1 DA 2`**

Set the scaling of the DFP (in display #1) to scaled mode.

setscreenposition

Earliest Driver 61.10

Effective with 70.80: nocommit, commit, and cancel options.

Description Set the TV or CRT screen position.

Format `rundll32.exe NvCpl.dll,dtcfg setscreenposition <display#> [<NV device moniker>] [<direction> <value> ["nocommit"]]|["commit"]|["cancel"]`

Where

- *direction* is either up, down, left, or right.
- *value* is the amount of relative screen movement, in pixels.
- "nocommit" (optional), specifies that the screen position change be in preview mode and not saved. If this option is not used, then the settings are saved.
- "commit" (optional), specifies that the settings made in preview mode be saved.
- "cancel" (optional), specifies that the settings made in preview mode be cancelled and that the previous settings be restored to the last saved setting.

Examples **`rundll32.exe NvCpl.dll,dtcfg setscreenposition 2,AA right 25 nocommit`**

Move the screen position on the first analog device on display 2 to the right 25 pixels, but show in preview mode and do not save.

`rundll32.exe NvCpl.dll,dtcfg setscreenposition 2,AA commit`

Save the screen position set in preview mode.

`rundll32.exe NvCpl.dll,dtcfg setscreenposition 2,AA cancel`

Cancel the screen position set in preview mode and restore to the last saved setting.

`rundll32.exe NvCpl.dll,dtcfg setscreenposition 2,AA right 25`

Move the screen position on the first analog device on display 2 to the right 25 pixels and save the setting.

settvformat

Earliest Driver 53.30

Description Set the TV format standard[i] on the specified display. This command adds custom timing parameters. When the advanced timing is no longer needed, they should be removed using the setdefault <timingmodesforcurrentmode> command.

New in version 61.20: *Setting HDTV formats on a DVI*.

- When setting an HDTV format on a TV connected to the DVI output, use the NV device moniker D*X* (see format below).
- For this command, the NV device monker D*X* works only for setting HD formats, and the format must be supported in the hardware.

Format

```
rundll32.exe NvCpl.dll,dtcfg settvformat
<display#> [<NV device moniker>] <TV format>
```

Where `TV format` is defined as follows:

- `0` : NTSC_M (analog TV)
- `1` : NTSC_J (analog TV)
- `2` : PAL_M (analog TV)
- `3` : PAL_A (analog TV)
- `4` : PAL_N (analog TV)
- `5` : PAL_NC (analog TV)
- `6` : HD480p (HDTV)
- `7` : HD720p (HDTV)
- `8` : HD1080i (HDTV)
- `9` : HD480i (HDTV)
- `10` : HD576i (HDTV)
- `11` : HD576p (HDTV)
- `D1` : (D-connector HDTV format—HD480i)
- `D2` : (D-connector HDTV format—HD480p)
- `D3` : (D-connector HDTV format—HD1080i)
- `D4` : (D-connector HDTV format—HD720p)
- `D5` : (D-connector HDTV format—HD1080p)

Example 1

```
rundll32.exe NvCpl.dll,dtcfg settvformat 1 0
                - or -
rundll32.exe NvCpl.dll,dtcfg settvformat TA 0
                - or -
rundll32.exe NvCpl.dll,dtcfg settvformat 1 TA 0
```

Set the TV standard (for display #1) to NTSC_M.

Example 2

```
rundll32.exe NvCpl.dll,dtcfg settvformat 1 D1
                - or -
rundll32.exe NvCpl.dll,dtcfg settvformat 1 8
```

Set the TV standard for display #1 to HD1080i format.

Note: This will not work if the associated connector is a DVI.

Example 3

```
rundll32.exe NvCpl.dll,dtcfg settvformat 1 DA 8
                - or -
rundll32.exe NvCpl.dll,dtcfg settvformat DA 8
```

Sets the HTDV standard for the DVI connector (for display #1) to HD1080i.

i. If the registry key NT5RestrictSelectedTVFormat (set in the INF) is present and set to 1, the allowable TV formats will depend on connector type.

setview

Description Set an nView multimonitor mode on the specified display device or devices.

For Dualview mode, <display#> determines the display numbering as well as the adapter as shown in the following example:

Example: "`setview <display#> dualview AA DA`"

If <display#> = 2, then AA will be 2 and DA will be 1.

If <display#> = 1, then AA will be 1 and DA will be 2.

The primary will be the first mnemonic = AA. Dualview should affect the primary only if the adapter(GPU) has a GDI primary, otherwise there will be no change in the primary.

Format

```
rundll32.exe NvCpl.dll,dtcfg setview <display#>
<viewtype> [<primary NV device monikerc>]
[<secondary NV device moniker>]
```

where

- *`viewtype`* can be any of the following:
  - `standard` (or `normal`)
  - `clone`
  - `hspan`
  - `vspan`
  - `dualview`
- *`primary NV device moniker`* and *`secondary NV device moniker`* indicate which display to assign as the primary and, in the case of clone or spanning mode, secondary devices. If these fields are not included, then the driver makes the assignments automatically.

Example 1 **`rundll32.exe NvCpl.dll,dtcfg setview 2 clone`**

Sets display #2 to Clone mode and let the driver assign the primary and secondary devices.

Example 2 **`rundll32.exe NvCpl.dll,dtcfg setview 2 clone AA`**

Sets display #2 to Clone mode using the first CRT found as the primary, and let the driver assign the secondary display device.

Example 3 **`rundll32.exe NvCpl.dll,dtcfg setview 2 clone AA DA`**

Sets display #2 to Clone mode using the first CRT found as the primary, and the first DFP found as the secondary display.

svof

Description Set the output format for the video signal.

Format `rundll32.exe NvCpl.dll,dtcfg svof <`*`display#`*`> <`*`format: 0,1,2,`*`>`

where format is one of

`0` : Auto select

`1`: Composite

`2`: S-video

Example **`rundll32.exe NvCpl.dll,dtcfg svof 2 0`**

Set the video format to "auto-select" on Windows display #2..

setsharpness

Earliest Driver `60.30`

Description Set the image sharpness.

Format `rundll32.exe NvCpl.dll,dtcfg setsharpness <`*`display#`*`> [<`*`NV device moniker`*`>] <`*`value: 0–21`*`>`

Where *`value`* is in the range 0–21.

Example **`rundll32.exe NvCpl.dll,dtcfg setsharpness 2 10`**

Set the image sharpness on Windows display #2 to a value ot 10.

set_normalize_sharpness

Earliest Driver `60.30`

Description Set the image sharpness, using normalized values.

Format `rundll32.exe NvCpl.dll,dtcfg set_normalize_sharpness <`*`display#`*`> [<`*`NV device moniker`*`>] <`*`value: 0.0–1.0`*`>`

Where *`value`* is in the normalized range 0.0–1.0.

Example **`rundll32.exe NvCpl.dll,dtcfg set_normalize_sharpness 2 0.5`**

Set the image shaprness on Windows display #2 to the normalized value of 0.5.

3

Control Panel APIs

The NVIDIA Display Control panel (**`nvcpl.dll`**) exports functions that allow you to configure your NVIDIA graphics card programmatically.

This chapter documents the following **`nvcpl.dll`** exported functions:

- "Desktop Configuration" on page 24
  - dtcfgex()
  - GetdtcfgLastError()
  - GetdtcfgLastErrorEx()
- "Display Information Functions" on page 30
  - NvGetDisplayInfo()
  - NvCplGetMSOrdinalDeviceString()
  - NvEnumDisplaySettings()
  - NvGetDisplayCustomName()
  - NvSetDisplayCustomName()
  - NvGetLastDisplaySettings()
  - NvGetDefaultDisplaySettings()
- "Gamma Ramp Functions" on page 44
  - NvColorGetGammaRamp()
  - NvColorSetGammaRamp()
  - NvColorGetGammaRampEx()
  - NvColorSetGammaRampEx()
- "Multi-Display Controls" on page 52
  - NvSelectDisplayDevice()
  - NvGetFullScreenVideoMirroringEnabled()
  - NvSetFullScreenVideoMirroringEnabled()
  - NvGetWindowsDisplayState()
- "Flat Panel Functions" on page 54

Desktop Configuration

dtcfgex()

`dtcfgex()` is a wrapper function for **`dtcfg`**, and calls **`dtcfg`** internally.

Function Prototype	`DWORD APIENTRY dtcfgex( IN OUT LPSTR lpszCmdLine);`
Parameters In	`LPSTR lpszCmdLine -- Supported dtcfg commands and arguments. See` "dtcfg" on page 6.
Return Values	The result of the **dtcfg** command, or the error.

In addition to the **`dtcfg`** command line functions, **`dtcfgex()`** can also use the following "get" functions that return a value to the `lpszCmdLine` buffer:

getdvcvalue

Description Get the Digital Vibrance level of the specified display.

Effective in driver version 61.40 (Rel60) /65.09 (Rel65): The "default" argument returns the defaul value.

Format

```
getdvcvalue <display#> [<NV device moniker>]
[default]
```

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine, "getdvcvalue 0");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the value of the digital vibrance level for the current primary display.

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

getcontrastvalue

Description Get the contrast level of the specified display.

Effective in driver version 61.40 (Rel60) /65.09 (Rel65): The "default" argument returns the defaul value.

Format `getcontrast <display#> [<NV device moniker>] <color channel> [default]`

Where *color channel* is one of the following:

- `red`
- `blue`
- `green`
- `all`

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine, "getcontrastvalue 0 all");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the contrast value of all color channels for the current primary display..

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

getdvihdformat

Earliest Driver `61.20`

Description Gets the list of high-definition formats supported in the DVI.

Format `getdvihdformat <display#> [<NV device moniker>]`

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine, "getdvihdformat 2");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the list of high-definition TV formats supported in the DVI.

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

get_normalize_contrastvalue

Description Get the contrast level of the specified display. This is an alternative to `getcontrastvalue`, and returns normalized values.

Effective in driver version 61.40 (Rel60) /65.09 (Rel65): The "default" argument returns the defaul value.

Format `getcontrast <display#> [<NV device moniker>] <color channel> [default]`

Where *`color channel`* is one of the following:

- `red`
- `blue`
- `green`
- `all`

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine, "getcontrastvalue 0 all");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the normalized (0.0–1.0) contrast value of all color channels for the current primary display.

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

getbrightnessvalue

Description Get the brightness level of the specified display.

Effective in driver version 61.40 (Rel60) /65.09 (Rel65): The "default" argument returns the defaul value.

Format `getbrightness <display#> [<NV device moniker>] <color channel> [default]`

Where *`color channel`* is one of the following:

- `red`
- `blue`
- `green`
- `all`

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine, "getbrightnessvalue 0 all");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the brightness value of all the color channels for the current primary display.

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

get_normalize_brightnessvalue

Description Get the brightness level of the specified display. This is an alternative to `getbrighntessvalue`, and returns normalized values.

Effective in driver version 61.40 (Rel60) /65.09 (Rel65): The "default" argument returns the defaul value.

Format `getbrightness <display#> [<NV device moniker>] <color channel> [default]`

Where *`color channel`* is one of the following:

- `red`
- `blue`
- `green`
- `all`

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine, "getbrightnessvalue 0 all");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the normalized (0.0–1.0) brightness value of all the color channels for the current primary display.

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

getgammavalue

Description Get the gamma level of the specified display.

Effective in driver version 61.40 (Rel60) /65.09 (Rel65): The "default" argument returns the defaul value.

Format `getgamma <display#> [<NV device moniker>] <color channel> [default]`

Where *`color channel`* is one of the following:

- `red`
- `blue`
- `green`
- `all`

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine, "getgammavalue 0 all");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the gamma value of all the color channels for the current primary display.

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

getsharpnessvalue

Earliest Driver `60.30`

Description Get the image sharpness value of the specified display.

If image sharpening is turned off or unavailable on the hardware, this function returns 0.

Effective in driver version 61.40 (Rel60) /65.09 (Rel65): The "default" argument returns the defaul value.

Format

```
getsharpnessvalue <display#> [<NV device moniker>]
[default]
```

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine, "getsharpnessvalue 0");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the sharpness value (0–21) of the current primary display.

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

get_normalize_sharpnessvalue

Earliest Driver `60.30`

Description Get the image sharpness value of the specified display. If image sharpening is turned off or unavailable on the hardware, this function returns 0.

This is an alternative to `getsharpnessvalue`, and returns normalized values.

Effective in driver version 61.40 (Rel60) /65.09 (Rel65): The "default" argument returns the defaul value.

Format

```
get_normalize_sharpnessvalue
        <display#> [<NV device moniker>] [default]
```

Example

```
char lpszCmdLine[50];
strcpy(lpszCmdLine,
        "get_normalize_sharpnessvalue 0");
dtcfgex(lpzCmdLine);
```

`lpszCmdLine` now contains the normalized (0.0–1.0) sharpness value of the current primary display.

See "Device Moniker Version 2 String Format" on page 99 for a description of the format to use for <NV device moniker>.

GetdtcfgLastError()

The function `GetdtcfgLastError()` returns the result of the last **`dtcg`** command.

Function Prototype	`DWORD WINAPI GetdtcfgLastError(void);`
Return Values	Result of the last **dtcfg** command. "No result available" upon failure.

GetdtcfgLastErrorEx()

The function `GetdtcfgLastErrorEx()` returns the result of the last **`dtcfg`** command and what that command was.

Function Prototype	`DWORD WINAPI GetdtcfgLastErrorEx`
	`( IN OUT LPSTR lpszCmdline,`
	`IN OUT DWORD *PdwCmdLineSize);`

Parameters In	`DWORD *PdwCmdLineSize --` The size of the buffer where the command string is to be returned.
Parameters Out	`LPSTR lpszCmdLine --` The last **dtcfg** command.
Return Values	Result of the last **dtcfg** command. "No result available" upon failure.

Display Information Functions

NvGetDisplayInfo()

The function `NvGetDisplayInfo()` returns detailed information for the specified display device.

See "NvGetDisplayInfo.c" on page 104 for an example program that demonstrates the use of this API.

Earliest Driver `56.50`

Function Prototype

```
BOOL APIENTRY NvGetDisplayInfo
                  ( IN LPCSTR pszUserDisplay,
                      OUT NVDISPLAYINFO* pDisplayInfo);
```

Parameters In `LPCSTR pszUserDisplay -- <device #> [NV device moniker]`

See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99.

Parameters Out `NVDISPLAYINFO* pDisplayInfo` -- Data where the first three fields determine the size and content of the remaining fields. See Table 3.1 for a description of each field.

Return Value **`TRUE`** on success

`FALSE` on failure.

Table 3.1 NVDISPLAYINFO Content

Data Field	Description / Bit Field for dwInputFields and dwOutputFields
`DWORD cbSize`	Size of the NVDISPLAYINFO structure (in bytes), set this field to sizeof(NVDISPLAYINFO) to indicate version level of structure
`DWORD dwInputFields1`	Specifies which members of the structure should be used on input to the function. Refer to individual bit fields for field masks `NVDISPLAYINFO1_`*`XXX`*. Set to `NVDISPLAYINFO1_ALL` for all fields.
`DWORD dwOutputFields1`	Specifies which members of the structure were processed as a result of the call. Refer to individual bit fields for field masks `NVDISPLAYINFO1_`*`XXX`*.
`DWORD dwInputFields2`	Specifies which members of the structure should be used on input to the function. Refer to individual bit fields for field masks `NVDISPLAYINFO2_`*`XXX`*. Set to `NVDISPLAYINFO1_ALL` for all fields.

Table 3.1 NVDISPLAYINFO Content

Data Field	Description / Bit Field for dwInputFields and dwOutputFields
`DWORD dwOutputFields2`	Specifies which members of the structure were processed as a result of the call. Refer to individual bit fields for field masks `NVDISPLAYINFO2_`*`XXX`*.
`char szWindowsDeviceName[_MAX_PATH]`	Device name for use with CreateDC (example: ".\\DISPLAY1") / `NVDISPLAYINFO1_WINDOWSDEVICENAME`
`char szAdapterName[MAX_NVDISPLAYNAME]`	User friendly name for the associated NVIDIA graphics card (example: GeForce FX 5200 Ultra) / `NVDISPLAYINFO1_ADAPTERNAME`
`char szDriverVersion[64]`	Display driver version string for the device (for example: "6.14.10.6003") / `NVDISPLAYINFO1_DRIVERVERSION`
`enum NVDISPLAYMODE nDisplayMode`	Display mode for head on the adapter -- `NVDISPLAYMODE_NONE`: No display, or unknown `NVDISPLAYMODE_STANDARD`: Single display mode `NVDISPLAYMODE_CLONE`: Clone mode `NVDISPLAYMODE_HSPAN`: H-Span mode `NVDISPLAYMODE_VSPAN`: V-Span mode `NVDISPLAYMODE_DUALVIEW`: Dualview mode `NVDISPLAYINFO1_DISPLAYMODE`
`DWORD dwWindowsMonitorNumber`	Windows monitor number for the adapter (numbers listed in Microsoft Display Panel) / `NVDISPLAYINFO1_WINDOWSMONITORNUMBER`
`int nDisplayHeadIndex`	Head index for the display on the adapter / `NVDISPLAYINFO1_DISPLAYHEADINDEX`
`BOOL bDisplayIsPrimary`	TRUE if the display head is primary on the adapter / `NVDISPLAYINFO1_DISPLAYISPRIMARY`
`char szDisplayName[MAX_NVDISPLAYNAME]`	User friendly name for the display device (for example: "EIZO L685") / `NVDISPLAYINFO1_DISPLAYNAME`
`char szVendorName[MAX_NVDISPLAYNAME]`	Vendor name for the display device, if available (for example: "EIZO") / `NVDISPLAYINFO1_VENDORNAME`

Table 3.1 NVDISPLAYINFO Content

Data Field	Description / Bit Field for dwInputFields and dwOutputFields
`char szModelName[MAX_NVDISPLAYNAME]`	Model name for the display device, if available (for example: "EIZ1728") / `NVDISPLAYINFO1_MODELNAME`
`char szGenericName[MAX_NVDISPLAYNAME]`	Generic name for the display device type (for example: "Digital Flat Panel") / `NVDISPLAYINFO1_GENERICNAME`
`DWORD dwUniqueId`	Unique identifier for the display device, including serial number. Zero if not available / `NVDISPLAYINFO1_UNIQUEID`

`enum NVDISPLAYTYPE nDisplayType`	Type of the display device --
	`NVDISPLAYTYPE_NONE`: No display, or unknown
	`NVDISPLAYTYPE_CRT`: CRT
	`NVDISPLAYTYPE_DFP`: DFP
	`NVDISPLAYTYPE_DFP_LAPTOP`: DFP Laptop
	`NVDISPLAYTYPE_TV`: TV
	`NVDISPLAYTYPE_TV_HDTV`: HDTV
	`NVDISPLAYTYPE_CLASS_MASK` : Mask for obtaining more specific information about the device class. If you perform
	`nDisplayType` & `NVDISPLAYTYPE_CLASS_MASK`
	then you get one of the above display type device classes. If you compare enumerated values against the nDisplayType field, then you get more detailed device subtype information.
	`NVDISPLAYINFO1_DISPLAYTYPE`

`DWORD mmDisplayWidth`	Width of the maximum visible display surface, or zero if unknown (in millimeters) / `NVDISPLAYINFO1_DISPLAYWIDTH`
`DWORD mmDisplayHeight`	Height of the maximum visible display surface, or zero if unknown (in millimeters) `NVDISPLAYINFO1_DISPLAYHEIGHT`
`float fGammaCharacteristic`	Gamma transfer characteristic for the monitor (for example: 2.2) / `NVDISPLAYINFO1_GAMMACHARACTERISTIC`
`DWORD dwOptimalPelsWidth`	Width of the display surface in optimal display mode (not necessarily highest resolution) / `NVDISPLAYINFO1_OPTIMALMODE`

Table 3.1 NVDISPLAYINFO Content

Data Field	Description / Bit Field for dwInputFields and dwOutputFields
`DWORD` `dwOptimalPelsHeight`	Height of the display surface in optimal display mode (not necessarily highest resolution) / `NVDISPLAYINFO1_OPTIMALMODE`
`DWORD` `dwOptimalDisplayFrequency`	Refresh frequency in optimal display mode (not necessarily highest resolution) / `NVDISPLAYINFO1_OPTIMALMODE`
`DWORD` `dwMaximumSafePelsWidth`	Width of the display surface in maximum safe display mode (not necessarily highest resolution). For DFPs, this is the native mode. / `NVDISPLAYINFO1_MAXIMUMSAFEMODE`
`DWORD` `dwMaximumSafePelsHeight`	Height of the display surface in maximum safe display mode (not necessarily highest resolution). For DFPs, this is the native mode./ `NVDISPLAYINFO1_MAXIMUMSAFEMODE`
`DWORD` `dwMaximumSafeDisplayFrequency`	Refresh frequency in maximum safe display mode (not necessarily highest resolution). For DFPs, this is the native mode. / `NVDISPLAYINFO1_MAXIMUMSAFEMODE`
`DWORD` `dwBitsPerPel`	Color resolution of the display device (for example: 8 bits for 256 colors) / `NVDISPLAYINFO1_BITSPERPEL`
`DWORD` `dwPelsWidth`	Width of the available display surface, including any pannable area (in pixels) / `NVDISPLAYINFO1_PELSWIDTH`
`DWORD` `dwPelsHeight`	Height of the available display surface, including any pannable area (in pixels) / `NVDISPLAYINFO1_PELSHEIGHT`
`DWORD` `dwDisplayFrequency`	Refresh frequency of the display device (in hertz) / `NVDISPLAYINFO1_DISPLAYFREQUENCY`
`RECT` `rcDisplayRect`	Desktop rectangle for the display surface (considers DualView and head offset) / `NVDISPLAYINFO1_DISPLAYRECT`
`DWORD` `dwVisiblePelsWidth`	Width of the visible display surface, excluding any pannable area (in pixels) / `NVDISPLAYINFO1_VISIBLEPELSWIDTH`
`DWORD` `dwVisiblePelsHeight`	Height of the visible display surface, excluding any pannable area (in pixels) / `NVDISPLAYINFO1_VISIBLEPELSHEIGHT`
`DWORD` `dwDegreesRotation`	Rotation angle of the display surface (in degrees) / `NVDISPLAYINFO1_DEGREESROTATION`

Table 3.1 NVDISPLAYINFO Content

Data Field	Description / Bit Field for dwInputFields and dwOutputFields
`enum` `NVTVFORMAT nTvFormat`	Television video signal format -- `NVTVFORMAT_NONE` : No format `NVTVFORMAT_NTSC_M` :NTSC/M `NVTVFORMAT_NTSC_J` : NTSC/J `NVTVFORMAT_PAL_M` : PAL/M `NVTVFORMAT_PAL_A` : PAL/B, D, G, H, I `NVTVFORMAT_PAL_N` : PAL/N `NVTVFORMAT_PAL_NC` : PAL/NC `NVTVFORMAT_HD480i` : HDTV 480i `NVTVFORMAT_HD480p` : HDTV 480p `NVTVFORMAT_HD576p` : HDTV 576p `NVTVFORMAT_HD720p` :HDTV 720p `NVTVFORMAT_HD1080i` : HDTV 1080i `NVTVFORMAT_HD1080p` : HDTV 1080p `NVTVFORMAT_HD576i` : HDTV 576i `NVTVFORMAT_HD720i` : HDTV 720i `NVDISPLAYINFO1_TVFORMAT`
`enum` `NVDFPSCALING nDfpScaling`	DFP scaling mode -- `NVDFPSCALING_NONE` : No scaling, or unknown `NVDFPSCALING_NATIVE` : Monitor scaling `NVDFPSCALING_SCALED` : Scaling `NVDFPSCALING_CENTERED` : Centering `NVDFPSCALING_SCALEDASPECT` : Scaling (Fixed aspect ratio) `NVDISPLAYINFO1_DFPSCALING`

`DWORD` `NVTVCONNECTORTYPES` `dwTVConnectorTypes`	Television connectors. The dwTVConnectorType field is zero if no physical TV connector exists on the board, non-zero if a physical TV connector exists.
	Individual bits indicate types of connections possible, but multiple bits can be set even if there is only one physical TV connector on board. For example, an S-Video connector may cause both Composite + S-Video bits to be set. --
	`NVTVCONNECTOR_UNKNOWN`
	`NVTVCONNECTOR_COMPOSITE`
	`NVTVCONNECTOR_SVIDEO`
	`NVTVCONNECTOR_COMPONENT`
	`NVTVCONNECTOR_EIAJ4120`
	`NVTVCONNECTOR_EIAJ4120CVBSBLUE`
	`NVTVCONNECTOR_SCART`
	
	`NVDISPLAYINFO1_TVCONNECTORTYPES`

Table 3.1 NVDISPLAYINFO Content

Data Field	Description / Bit Field for dwInputFields and dwOutputFields

`DWORD`
`NVCURRENTCONNECTORTYPE`
`dwCurrentConnectorType`
	Television active connector (values not enumerated).
	`NVDISPLAYINFO1_CURRENTCONNECTORTYPE`

`DWORD`
`NVBOARDTYPE dwBoardType`
	Effective in driver version 65.07
	Type of board, such as Quadro, NVS, etc.
	`NVBOARDTYPE_GEFORCE` : GeForce board
	`NVBOARDTYPE_QUADRO` : Quadro board
	`NVBOARDTYPE_NVS` : NVS board
	`NVDISPLAYINFO1_BOARDTYPE`

`DWORD` `dwDisplayInstance`	*Effective in driver version 65.80* Display instance number (instance of szDisplayName) or zero if indeterminant. `NVDISPLAYINFO1_DISPLAYINSTANCECOUNT`
`DWORD` `dwDisplayInstanceCount`	*Effective in driver version 65.80* Display instance count (instances of szDisplayName) or zero if indeterminant. `NVDISPLAYINFO1_DISPLAYINSTANCECOUNT`

`char`
`szProductName[MAX_NVDISPLAYNAME]`
	Effective in driver version 65.80
	Product name for display device if available, bypasses user customization of szDisplayName (for example: "EIZO L685").
	`NVDISPLAYINFO2_PRODUCTNAME`

`BOOL` `bDVIOverHDTV`	*Effective in driver version 66.00* HDTV video using the DVI connector (“HDTV-over-DVI”). `NVDISPLAYINFO2_DVIOVERHDTV`

`char`
`szConnectedMoniker`
`[MAX_NVMONIKER]`
	Effective in driver version 66.50
	Device moniker for display based on physically connected devices (empty if not connected).
	`NVDISPLAYINFO2_CONNECTEDMONIKER`

`char`
`szActiveMoniker`
`[MAX_NVMONIKER]`
	Effective in driver version 66.50
	Device moniker for display based on active display outputs (e.g. those attached to desktop, empty if not attached).
	`NVDISPLAYINFO2_ACTIVEMONIKER`

`BOOL` `bSLIEnabled`	*Effective in driver version 66.80* SLI is turned on and active. `NVDISPLAYINFO2_SLIENABLED`

Table 3.1 NVDISPLAYINFO Content

Data Field	Description / Bit Field for dwInputFields and dwOutputFields
`BOOL` `bSLIConnector`	*Effective in driver version 66.80* SLI connector exists `NVDISPLAYINFO2_SLICONNECTOR`
`BOOL` `bSLICapable`	*Effective in driver version 66.80* SLI can be enabled for this display. `NVDISPLAYINFO2_SLICAPABLE`

`char`
`szCustomName`
`[MAX_NVDISPLAYNAME]`
	Effective in driver version 70.20
	Custom name for display device type, if available (for example: "OEM1 Monitor on Left")
	`NVDISPLAYINFO2_CUSTOMNAME`

NvCplGetMSOrdinalDeviceString()

The function returns in a buffer the moniker strings for the Microsoft ordinal device passed in. In the case of Span/Clone modes, it returns two monikers (primary then secondary). In the case of Dualview/Single, it passes back the moniker for the Microsoft device.

The monikers are comma-separated and conform to the device moniker specification (see "Device Moniker Version 2 String Format" on page 99).

The function can only detect attached displays. If you pass in an unattached display ordinal, it will fail.

Earliest Driver	`65.60`
Function Prototype	`BOOL APIENTRY NvCplGetMSOrdinalDeviceString (IN DWORD dwMSOrdinal, OUT LPSTR lpszTextBuffer, IN DWORD cbTextBuffer );`
Parameters In	`dwMSOrdinal` -- The microsoft display ordinal number `cbTextBuffer` -- The size of the lpszTextBuffer
Parameters Out	`lpszTextBuffer` -- A buffer (must be at least 6 characters in length) that contains the returned comma-separated moniker strings.
Return Values	`True` on success.

NvEnumDisplaySettings()

The function `NvEnumDisplaySettings()` enumerates available display resolutions for a specified nView display mode and combination of devices. This function allows for resolutions to be filtered against the capabilities of the display device, or can be used to return a complete list resolutions supported by the graphics card.

When queried using standard (single-display) nView display mode, this filtering allows physical device resolutions to be retrieved. To produce a non-panning resolutions list, intersect the physical device resolutions for each display head with the available resolutions returned for the target mode (Clone, Span).

The display resolutions available on a display device in standard (single-display) mode may differ from those available for that display device in Clone or Span modes.

Currently, mode enumeration under DualView mode is not supported.

See “NvGetDisplayInfo.c” on page 104 for an example program that demonstrates the use of this API.

Earliest Driver `66.41`

Function Prototype

```
NVRESULT NVAPIENTRY NvEnumDisplaySettings
         ( IN LPCSTR pszUserDisplay,
           IN NVDISPLAYMODE displayMode,
           IN DWORD dwDevModeSize,
           OUT DEVMODE* pDevModes,
           IN OUT DWORD* pdwNumDevModes,
            IN DWORD dwFlags );
```

Parameters In

`LPCSTR pszUserDisplay` --

See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99.

`NVDISPLAYMODE displayMode --`

Display mode for display resolution enumeration. EnumDisplaySettings supports the following modes:

`NVDISPLAYMODE_STANDARD`

`NVDISPLAYMODE_CLONE`

`NVDISPLAYMODE_HSPAN`

`NVDISPLAYMODE_VSPAN`

`DEVMODE* pDevModes --`

Pointer to caller allocated buffer to receive display resolutions (can be NULL)

`DWORD cbSizeDevMode --`

Size of DEVMODE structure referenced by pDevModes

`DWORD* pdwNumDevModes --`

Pointer to number of pDevModes elements allocated by caller

`DWORD dwFlags --` Flags for display mode enumeration:

0: filter the display resolutions against the capabilities of the display devices.

`EDS_RAWMODE` : report modes supported by adapter regardless of monitor capabilities

Parameters Out

`DWORD* pdwNumDevModes --`

Pointer to number of pDevModes elements enumerated by function.

`DEVMODE* pDevModes --`

Pointer to enumerated display resolutions (can be NULL)

NvEnumDisplaySettings updates the following five fields of each element:

`dmBitsPerPel`

`dmPelsWidth`

`dmPelsHeight`

`dmDisplayFlags`

`dmDisplayFrequency`

Return Values

`NV_OK` - Success

`NV_OUTOFMEMORY` - Supplied pDevModes buffer is too small (see returned pdwNumDevModes for size requirement)

`NV_ACCESSDENIED` - Could not access specified device moniker in requested display mode

---

The structure DEVMODE is defined in the Win32 documentation.

NvGetDisplayCustomName()

The function `NvGetDisplayCustomName()` returns the custom monitor name for the specified display device. If a custom monitor name has not been established for the specified display device, this function returns failure.

Earliest Driver	`70.20`
Function Prototype	`NVRESULT NVAPIENTRY NvGetDisplayCustomName ( IN LPCSTR pszUserDisplay, OUT LPSTR pszTextBuffer, IN DWORD cbTextBuffer );`
Parameters In	`LPCSTR pszUserDisplay --` See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99. `DWORD cbTextBuffer --` Size of buffer to receive custom monitor name in bytes.
Parameters Out	`LPSTR pszTextBuffer --` Pointer to buffer receive custom monitor name (maximum length will be MAX_NVDISPLAYNAME).
Return Values	**`NV_OK`** - Success **`NV_NOTFOUND`** - Could not find custom monitor name for specified display device. **`NV_OUTOFMEMORY`** - Supplied user buffer is too small for custom monitor name. else - Failure

NvSetDisplayCustomName()

The function `NvSetDisplayCustomName()` sets the custom monitor name for the specified display device. To clear the custom monitor name for display device, specify an empty custom monitor name string.

Earliest Driver	`70.20`
Function Prototype	`NVRESULT NVAPIENTRY NvSetDisplayCustomName ( IN LPCSTR pszUserDisplay, IN LPCSTR pszTextBuffer );`
Parameters In	`LPCSTR pszUserDisplay --` See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99. `LPCSTR pszTextBuffer --` Pointer to custom monitor name string (maximum length is MAX_NVDISPLAYNAME).
Parameters Out	N/A
Return Values	**`NV_OK`** - Success else - Failure

NvGetLastDisplaySettings()

The function `NvGetLastDisplaySettings()` returns the last saved display resolutions for a specified nView display mode and combination of devices. This function will return failure if the specified display configuration has not been previously enabled.

Earliest Driver `66.80`

Function Prototype

```
NVRESULT NVAPIENTRY NvGetLastDisplaySettings
             ( IN LPCSTR pszUserDisplay,
               IN NVDISPLAYMODE displayMode,
               OUT DEVMODE* pDevMode,
                IN DWORD dwFlags );
```

Parameters In

`DWORD pszUserDisplay --`

See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99.

`NVDISPLAYMODE displayMode --`

Display mode for display resolution enumeration.

NvEnumDisplaySettings supports the following modes:

```
NVDISPLAYMODE_STANDARD
NVDISPLAYMODE_CLONE
NVDISPLAYMODE_HSPAN
NVDISPLAYMODE_VSPAN
NVDISPLAYMODE_DUALVIEW
```

`DWORD dwFlags --`

HIWORD(dwFlags): Head number for DualView
(0 is first device in moniker pair, 1 is second device in moniker pair)

LOWORD(dwFlags): Flags for display mode enumeration:

`EDS_ALLMODES` reports modes regardless of adapter or monitor capabilities

`EDS_RAWMODE` reports the modes supported by adapter regardless of monitor capabilities

If the current display mode is DualView, this function will fail unless EDS_ALLMODES is used

Parameters Out `DEVMODE* pDevMode` -- Pointer to display resolution.

Return Values

`NV_OK` - Success

`NV_NOTFOUND` - Could not find last display mode for specified display mode and display combination.

`NV_ACCESSDENIED` - Could not access specified device moniker in requested display mode.

NvGetDefaultDisplaySettings()

The function `NvGetDefaultDisplaySettings()` returns the default display resolutions for a specified nView display mode and combination of devices. This function returns the display resolutions that would be used when there is no last saved resolution information returned by `NvGetLastDisplaySettings()`.

Earliest Driver 66.80

Function Prototype

```
NVRESULT NVAPIENTRY NvGetDefaultDisplaySettings
                   ( IN LPCSTR pszUserDisplay,
                     IN NVDISPLAYMODE displayMode,
                     OUT DEVMODE* pDevMode,
                       IN DWORD dwFlags );
```

Parameters In

`DWORD pszUserDisplay --`

See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99.

NVDISPLAYMODE displayMode --

Display mode for display resolution enumeration.

NvEnumDisplaySettings supports the following modes:

```
NVDISPLAYMODE_STANDARD
NVDISPLAYMODE_CLONE
NVDISPLAYMODE_HSPAN
NVDISPLAYMODE_VSPAN
NVDISPLAYMODE_DUALVIEW
```

`DWORD dwFlags --`

HIWORD(dwFlags): Head number for DualView
(0 is first device in moniker pair, 1 is second device in moniker pair)

LOWORD(dwFlags): Flags for display mode enumeration:

`EDS_ALLMODES` reports modes regardless of adapter or monitor capabilities

`EDS_RAWMODE` report modes supported by adapter regardless of monitor capabilities

If the current display mode is DualView, this function will fail unless EDS_ALLMODES is used

Parameters Out

`DEVMODE* pDevMode`

`Pointer to display resolution`

Return Values

`NV_OK` - Success

`NV_NOTFOUND` - Could not find last display mode for specified display mode and display combination.

`NV_ACCESSDENIED` - Could not access specified device moniker in requested display mode.

Gamma Ramp Functions

The Gamma Ramp API provides functions that read and write the gamma values for the GPU. The following functions that are exported from `nvcpl.dll`:

NvColorGetGammaRamp()

This function has been deprecated, but is available for legacy support. New applications should use* NvColorGetGammaRampEx()*.

`NvColorGetGammaRamp()` gets the current gamma color values.

Function Prototype	`BOOL CDECL NvColorGetGammaRamp ( IN LPCSTR pszUserDisplay, OUT GAMMARAMP* pGammaRamp);`
Parameters In	`LPCSTR pszUserDisplay` -- Either specify "all" for all displays, or specify a particular display using the device moniker format. See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99.
Parameters Out	`GAMMARAMP* pGammaRamp` -- the current gamma table values
Return Values	**TRUE** if the gamma values have been retrieved. **FALSE** if the retrieval failed.

NvColorSetGammaRamp()

This function has been deprecated, but is available for legacy support. New applications should use* NvColorSetGammaRampEx()*.

`NvColorSetGammaRamp()` sets the gamma color values.

Function Prototype	`BOOL CDECL NvColorSetGammaRamp ( IN LPCSTR pszUserDisplay, IN DWORD dwUserRotateFlag, IN const GAMMARAMP* pGammaRamp);`
Parameters In	`LPCSTR pszUserDisplay` -- Either specify "all" for all displays, or specify a particular display using the device moniker format. See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99. `DWORD dwUserRotateFlag` -- display rotation flag `GAMMARAMP* pGammaRamp` -- the new gamma table values
Return Values	**TRUE** if the new gamma values have been applied. **FALSE** otherwise, for the following reasons: • The display name is not valid. • The gamma values do not produce a valid gamma ramp.

Sample Code Using GammaRamp

The following is an example of how to use the GammaRamp APIs:

```
    // Single Head Display Gamma Support
    typedef struct GAMMARAMP {
        WORD   wRed  [256];
        WORD   wGreen[256];
        WORD   wBlue [256];
    } GAMMARAMP, *PGAMMARAMP;



typedef struct GAMMARAMP
{
  WORD  wRed  [256];
  WORD  wGreen[256];
  WORD  wBlue [256];
} GAMMARAMP, *PGAMMARAMP;

typedef BOOL (*PCOLORSETGAMMARAMP)( LPTSTR, DWORD, PGAMMARAMP );
typedef BOOL (*PCOLORGETGAMMARAMP)( LPTSTR, PGAMMARAMP );



void main()
{
  HINSTANCE           hCpl = NULL;
  PCOLORGETGAMMARAMP  pGetGamma = NULL;
  PCOLORSETGAMMARAMP  pSetGamma = NULL;
  GAMMARAMP           Gamma;

  memset( &Gamma, 0, sizeof(Gamma) );

  // Load the NVIDIA control panel applet. This from where the
  //  gamma functions are exported.
  hCpl = LoadLibrary( "nvcpl.dll" );
  if( hCpl == NULL )
  {
      return;
  }
  pGetGamma = (PCOLORGETGAMMARAMP)GetProcAddress( hCpl, 
"NvColorGetGammaRamp" );
  if( pGetGamma == NULL )
  {
      FreeLibrary( hCpl );
      return;
  }

  // Retrieve the gamma table.
  pGetGamma( "a0", &Gamma );

  for( int i = 0; i < 256; i++ )
  {
      // Do something with gamma values...
      //Gamma.wRed[ i ] = ...;
      //Gamma.wGreen[ i ] = ...;
      //Gamma.wBlue[ i ] = ...;
  }

  pSetGamma = (PCOLORSETGAMMARAMP)GetProcAddress( hCpl, 
"NvColorSetGammaRamp" );
  if( pSetGamma == NULL )
  {
      FreeLibrary( hCpl );
      return;
  }

  // Set the new gamma values.
  pSetGamma( "a0", 0xFFFFFFFF, &Gamma );

  FreeLibrary( hCpl );
}
```

NvColorGetGammaRampEx()

`NvColorGetGammaRampEx()` gets the current gamma color values for the desktop, overlays, and full-screen videos.

Earliest Driver	`60.30`
Function Prototype	`BOOL APIENTRY NvColorGetGammaRampEx (IN LPCSTR pszUserDisplay, OUT GAMMARAMP* pGammaRamp, IN NVCOLORAPPLY applyFrom);`
Parameters In	`LPCSTR pszUserDisplay --` Either specify "all" for all displays, or specify a particular display using the device moniker format. See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99. `NVCOLORAPPLY applyFrom` -- specifies from where to get the gamma settings:
Parameters Out	`GAMMARAMP* pGammaRamp --` the current gamma table values
Return Values	**`TRUE`** if the gamma values have been retrieved. **`FALSE`** if the retrieval failed.

```
BOOL APIENTRY NvColorGetGammaRampEx
              (IN LPCSTR pszUserDisplay,
               OUT GAMMARAMP* pGammaRamp,
                IN NVCOLORAPPLY applyFrom);
```

```
enum NVCOLORAPPLY
{
  NVCOLORAPPLY_DESKTOP,           //  Desktop
  NVCOLORAPPLY_OVERLAYVMR,        //Overlay/Video Mirroring
  NVCOLORAPPLY_FULLSCREENVIDEO,   //  Fullscreen Video
  NVCOLORAPPLY_COUNT   //  Number of color settings targets
};
```

NvColorSetGammaRampEx()

`NvColorSetGammaRampEx()` sets the current gamma color values for the desktop, overlays, or full-screen videos.

Earliest Driver	`60.30`
Function Prototype	`BOOL APIENTRY NvColorSetGammaRampEx (IN LPCSTR szUserDisplay, IN const GAMMARAMP* pGammaRamp, IN NVCOLORAPPLY applyTo);`

Parameters In	`LPCSTR szUserDisplay` -- Either specify "all" for all displays, or specify a particular display using the device moniker format.
	See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99.
	`NVCOLORAPPLY applyTo` -- specifies where to apply the gamma settings:
	`enum NVCOLORAPPLY`
	`{`
	`NVCOLORAPPLY_DESKTOP,` // Desktop
	`NVCOLORAPPLY_OVERLAYVMR,` //Overlay/Video Mirroring
	`NVCOLORAPPLY_FULLSCREENVIDEO,` // Fullscreen Video
	`NVCOLORAPPLY_COUNT` // Number of color settings targets
	`};`
	`const GAMMARAMP* pGammaRamp --` the current gamma table values.

Return Values	**`TRUE`** if the gamma values have been applied. **`FALSE`** if `NVCOLORAPPLY` does not allow gamma settings to be changed.

Sample Code Using GammaRampEx

```
// GammaRamp.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <windows.h>

typedef struct _GAMMARAMP {
        WORD  Red[256];
        WORD  Green[256];
        WORD  Blue[256];
} GAMMARAMP, *PGAMMARAMP;
```

```
enum NVCOLORAPPLY
{
 NVCOLORAPPLY_DESKTOP,         // Apply color settings to Desktop
 NVCOLORAPPLY_OVERLAYVMR,      // Apply color settings to
                               //Overlay/Video Mirroring
 NVCOLORAPPLY_FULLSCREENVIDEO, // Apply color settings to
                               //Fullscreen Video

 NVCOLORAPPLY_COUNT            // Number of apply color settings targets
};

typedef BOOL (APIENTRY* fNvColorGetGammaRampEx)( LPCSTR szUserDisplay,
               PGAMMARAMP pGammaNew, NVCOLORAPPLY applyFrom );
typedef BOOL (APIENTRY* fNvColorSetGammaRampEx)( LPCSTR szUserDisplay,
               const PGAMMARAMP pGammaNew, NVCOLORAPPLY applyTo );

int main(int argc, char* argv[])
{
 // Parse command-line arguments
 if (argc != 2)
 {
     fprintf(stderr, "usage: %s <colorapply>\n"
                   "  where colorapply is { desktop, overlay,
                        fullscreenvideo }\n", argv[0]);
     return 0;
 }
 NVCOLORAPPLY colorApply    = NVCOLORAPPLY_DESKTOP;
 char*        pszColorApply = "???";

 if (!stricmp(argv[1], "desktop"))
 {
     colorApply    = NVCOLORAPPLY_DESKTOP;
     pszColorApply = "desktop";
 }
 else if (!stricmp(argv[1], "overlay"))
 {
     colorApply    = NVCOLORAPPLY_OVERLAYVMR;
     pszColorApply = "overlay";
 }
 else if (!stricmp(argv[1], "fullscreenvideo"))
```

```
{
    colorApply    = NVCOLORAPPLY_FULLSCREENVIDEO;
    pszColorApply = "fullscreenvideo";
}
else
{
    fprintf(stderr, "Invalid color apply argument: \"%s\"\n", argv[1]);
    return 1;
}

// Load control panel library
HMODULE hLib = LoadLibrary("nvcpl.dll");
if (hLib == NULL)
{
    fprintf(stderr, "Failed to load library.\n");
    return 2;
}

// Bind to gamma ramp functions
fNvColorGetGammaRampEx pfnNvColorGetGammaRampEx =
       (fNvColorGetGammaRampEx) GetProcAddress(hLib,
                                    "NvColorGetGammaRampEx");

fNvColorSetGammaRampEx pfnNvColorSetGammaRampEx =
       (fNvColorSetGammaRampEx) GetProcAddress(hLib,
                                    "NvColorSetGammaRampEx");

if ((pfnNvColorGetGammaRampEx == NULL) ||
     (pfnNvColorSetGammaRampEx == NULL))
{
    fprintf(stderr, "Failed to bind to gamma ramp functions.\n");
    FreeLibrary(hLib);
    return 3;
}

// Get gamma ramp
GAMMARAMP gammaRamp = {0};
if (!pfnNvColorGetGammaRampEx("AA", &gammaRamp, colorApply))
{
        fprintf(stderr, "Failed to get gamma ramp for %s.\n",
                  pszColorApply);
        FreeLibrary(hLib);
        return 4;
    }

    // Invert gamma ramp
   #define SWAP(x,y) { WORD t = (x); (x) = (y); (y) = (t); }
    for (int i = 0;
                i < 256/2; i++)
    {

        SWAP(gammaRamp.Red  [i], gammaRamp.Red  [255-i]);
        SWAP(gammaRamp.Green[i], gammaRamp.Green[255-i]);
        SWAP(gammaRamp.Blue [i], gammaRamp.Blue [255-i]);
    }

    // Set gamma ramp
    if (!pfnNvColorSetGammaRampEx("AA", &gammaRamp, colorApply))
    {
        fprintf(stderr, "Failed to set gamma ramp for %s.\n",
                  pszColorApply);
        FreeLibrary(hLib);
        return 5;
    }

    fprintf(stderr, "Inverted gamma ramp for %s.\n", pszColorApply);
    return 0;
   }
```

Multi-Display Controls

NvSelectDisplayDevice()

The function `NvSelectDisplayDevice()` is used primarily to select a particular adapter in a multiple adapter system. This is accomplished by specifying one of the display numbers from the Windows Settings page. The resulting adapter becomes the default for subsequent function calls, but an individual call can override this if it specifies a different display number.

See “NvGetDisplayInfo.c” on page 104 for an example program that demonstrates the use of this API.

Earliest Driver	`56.50`
Function Prototype	`BOOL CDECL NvSelectDisplayDevice (IN UINT nWindowsMonitorNumber);`
Parameter In	`UINT nWindowsMonitorNumber --` the display number shown on the Windows Display Properties->Settings page.
Return Values	**`TRUE`** on success **`FALSE`** on failure.

NvGetFullScreenVideoMirroringEnabled()

The function `NvGetFullScreenVideoMirroringEnabled()` returns whether the full screen video mirroring is enabled for the specified display.

Function Prototype	`BOOL CDECL NvGetFullScreenVideoMirroringEnabled (IN LPCSTR pszUserDisplay, OUT BOOL* pbEnabled);`
Parameter In	`LPCSTR pszUserDisplay --` Either specify “all” for all displays, or specify a particular display using the device moniker format. See “Passing Device Monikers in pszUserDisplay” on page 100 as well as “Device Moniker Version 2 String Format” on page 99.
Parameters Out	`BOOL* pbEnabled:` **`TRUE`** if full-screen video mirroring is enabled. **`FALSE`** if full-screen video mirroring is disabled.
Return Values	**`TRUE`** on success **`FALSE`** on failure.

NvSetFullScreenVideoMirroringEnabled()

The function `NvSetFullScreenVideoMirroringEnabled()` enables or disables full-screen video mirroring on the specified display.

Function Prototype	`BOOL CDECL NvSetFullScreenVideoMirroringEnabled (IN LPCSTR pszUserDisplay, IN BOOL pbEnabled);`
Parameters In	`LPCSTR pszUserDisplay --` Either specify “all” for all displays, or specify a particular display using the device moniker format. See “Passing Device Monikers in pszUserDisplay” on page 100 as well as “Device Moniker Version 2 String Format” on page 99. `BOOL pbEnabled --` **`TRUE`** to enable full-screen video mirroring **`FALSE`** to disable disable full-screen video mirroring.
Return Values	**`TRUE`** on success **`FALSE`** on failure.

NvGetWindowsDisplayState()

This function has been deprecated, but is available for legacy support. New applications should use* NvGetDisplayInfo()*.

The function `NvGetWindowsDisplayState()` returns the multimonitor state for the specified Windows display.

Function Prototype	`int NVAPIENTRY NvGetWindowsDisplayState ( IN UINT nWindowsMonitorNumber );`
Parameters In	`nWindowsMonitorNumber --` the display number shown on the Windows Display Properties->Settings page. A value of 0 indicates the current Windows primary display.
Return Values	`NVGWDS_VIEW_UNKNOWN` -1 : Unknown state or view mode.
	`NVGWDS_FAILED` 0 : The call failed— internal error
	`NVGWDS_NOT_FOUND` 1 : Unrecognized Windows monitor number
	`NVGWDS_UNATTACHED` 2 : Graphics card not attached to desktop
	`NVGWDS_ATTACHED` 3 : Graphics card attached to desktop but not an NVIDIA device
	`NVGWDS_STANDARD` 4 : Graphics card in Single-Display mode (not in DualView)
	`NVGWDS_DUALVIEW` 5 : Graphics card in DualView mode (not in Single-Display mode)
	`NVGWDS_CLONE` 6 : Graphics card in Clone mode
	`NVGWDS_HSPAN` 7 : Graphics card in Horizontal Span mode
	`NVGWDS_VSPAN` 8 : Graphics card in Vertical Span mode

Flat Panel Functions

NvCplGetFlatPanelNativeRes()

This function has been deprecated, but is available for legacy support. New applications should use* NvGetDisplayInfo()*.

The function `NvCplGetFlatPanelNativeRes()` returns the maximum or native resolution of the digital flat panel.

Function Prototype	`BOOL APIENTRY NvCplGetFlatPanelNativeRes ( IN LPCSTR pszUserDisplay OUT DWORD *pdwHorizontalPixels OUT DWORD *pdwVerticalPixels );`
Parameters In	`pszUserDisplay --` Specifies the device to check. See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99. `*pdwHorizontalPixels` -- Pointer to the DWORD to place the maximum width data. `*pdwVerticalPixels --` Pointer to the DWORD to place the maximum height data.
Return Values	**`TRUE`** if the resolution is obtained successfully.

NvCplGetScalingStatus()

This function has been deprecated, but is available for legacy support. New applications should use* NvGetDisplayInfo()*.

The function `NvCplGetScalingStatus()` returns the flat panel scaling. .

Function Prototype	`BOOL CDECL NvCplGetScalingStatus ( IN LPCSTR pszUserDisplay, OUT DWORD* pdwScalingMode );`
Parameters In	`pszUserDisplay --` Specifies the device to check. See "Passing Device Monikers in pszUserDisplay" on page 100 as well as "Device Moniker Version 2 String Format" on page 99.
Parameter Out	`DWORD* pdwScalingMode` -- • `0x00000000` : Default • `0x00000001`: Native • `0x00000002`: Scaled • `0x00000003`: Centered • `0x00000005`: Aspect scaling (for wide panel LCD)
Return Values	**`TRUE`** on success. **`FALSE`** on failure.

Connection Information

NvCplRefreshConnectedDevices()

The function `NvCplRefreshConnectedDevices()` refreshes the connection state cache for all display outputs on the selected GPU (see "NvSelectDisplayDevice()" on page 52). The basic operation involves performing an exhaustive device scan and then deactivating any active display outputs that do not have a device physically connected.

This detection routine may produce noticable flashes on some display devices.

See "NvGetDisplayInfo.c" on page 104 for an example program that demonstrates the use of this API..

Earliest Driver	`65.93`

Function Prototype	`BOOL NVAPIENTRY NvCplRefreshConnectedDevices`
	`( IN DWORD dwFlags );`

Parameters In	`DWORD dwFlags --`
	Flags bitmask for modifying the basic operation.
	`NVREFRESH_NONINTRUSIVE 0x00000001`
	Performs less exhaustive search and does not detach any active display heads which have been physically disconnected since detection.
	`NVREFRESH_SYSTEMWIDE   0x00000002`
	Performs refresh for all graphics adapters. If not specified, the selected graphics adapter is used.

Return Values	**`TRUE`** : Success. **`FALSE`**: Failure

NvCplGetRealConnectedDevicesString()

The function `NvCplGetRealConnectedDevicesString()` returns a list of all the displays that are connected to the system. You can specify whether to return only the displays that are active, or all connected displays

This function is the recommended method to get device monikers for connected devices. It replaces the `NvCplGetConnectedDeviceString()` function.

The display strings are comma-separated and follow the NVIDIA device moniker format for connected devices (prefixed with a "#"). See "Device Moniker Version 2 String Format" on page 99

See "NvGetDisplayInfo.c" on page 104 for an example program that demonstrates the use of this API. .

Earliest Driver `66.60`

Function Prototype

```
BOOL NVAPIENTRY
     NvCplGetRealConnectedDevicesString
          ( OUT LPSTR lpszTextBuffer,
              IN DWORD cbTextBuffer,
               IN BOOL bOnlyActive );
```

Parameters In `cbTextBuffer --` The size of the 'receive' buffer.

`bOnlyActive --`

FALSE to return all the connected devices.

TRUE to return only the connected devices that are active.

Parameter Out `lpszTextBuffer --` The buffer to receive the requested strings.

Return Values **`TRUE`** on success.

`FALSE` on failure.

NvCplGetConnectedDevicesString()

This function has been deprecated, but is available for legacy support.

It will not work in some cases where one or more connected devices are not actively being driven. New applications should use the function `NvCplGetRealConnectedDevicesString()` instead to retrieve more accurate results.

The function `NvCplGetConnectedDevicesString()` returns a list of all the displays that are connected to the system. You can specify whether to return only the displays that are active, or all connected displays

The display strings are comma-separated and follow the NVIDIA device moniker format. See "Device Moniker Version 2 String Format" on page 99.

Function Prototype

```
BOOL NVAPIENTRY NvCplGetConnectedDevicesString
      ( OUT LPSTR lpszTextBuffer,
        IN DWORD cbTextBuffer,
         IN BOOL bOnlyActive );
```

Parameters In

`cbTextBuffer --` The size of the 'receive' buffer.

`bOnlyActive --`

FALSE to return all the connected devices.

TRUE to return only the connected devices that are active.

Parameter Out

`lpszTextBuffer --` The buffer to receive the requested comma-separated strings.

Return Values

`TRUE` on success.

`FALSE` on failure.

NvCplGetActiveDevicesString()

The function `NvCplGetActiveDevicesString` returns the comma-delimited device monikers string for all active display outputs on the selected GPU. Unlike `NvCplConnectedDevicesString`, active display outputs with no device physically attached are not filtered from the results.

This function does not detect the devices, but rather uses the connected device state that was cached by the driver during such system events as bootup, logon, or opening of the display properties control panel, thereby avoiding the screen flashes associated with device detection. To refresh the cached connector state prior to calling this routine, use `NvCplRefreshConnectedDevices()` with the `NVREFRESH_NONINTRUSIVE` flag.

See "NvGetDisplayInfo.c" on page 104 for an example program that demonstrates the use of this API.

Earliest Driver	`65.93`
Function Prototype	`BOOL NVAPIENTRY NvCplGetActiveDevicesString (OUT LPSTR lpszTextBuffer, IN DWORD cbTextBuffer );`
Parameters In	`DWORD cbTextBuffer` -- Size of caller-supplied text buffer in bytes.
Parameter Out	`LPSTR lpszTextBuffer` -- Pointer to caller-supplied text buffer to receive device monikers string.
Return Values	**`TRUE`** : Success. **`FALSE`**: Failure

The display strings follow the NVIDIA device moniker format for active devices (see "Device Moniker Version 2 String Format" on page 99).

Note: Device monikers for active but disconnected displays will be prefixed with a minus sign (for example, "-AB"). These monikers can be used with most other API functions that do not reference the display directly. API functions such as `NvGetDisplayInfo()` support this type of moniker, but functions such as `NvGetDisplayCustomName()` do not.

NvGetPhysicalConnectorInfo()

The function `NvGetPhysicalConnectorInfo()` returns information about the physical connectors on the graphics card. This function reads information from an optional block of the NV4X and higher graphics card video BIOS. When used with graphics cards which do not have this optional block, the function will return NV_NOTSUPPORTED..

Earliest Driver `65.91`

Function Prototype

```
NVRESULT NVAPIENTRY NvGetPhysicalConnectorInfo
                (IN UINT nAdapterNumber,
                 OUT NVCONNECTORINFO* pConnectorInfo );
```

Parameters In `UINT nAdapterNumber --`

Windows display number for the graphics card.

Parameter Out `NVCONNECTORINFO* pConnectorInfo` --

Pointer to receive the physical connector information. Caller should initialize pConnectorInfo->cbSize to sizeof(NVCONNECTORINFO).

Return Values **`NV_OK`** : Success.

`NV_NOTSUPPORTED:` Unsupported feature (requires compatible video BIOS).

Table 3.2 NVCONNECTORINFO Content

Datas	Description
`DWORD cbSize;`	Size of the `NVCONNECTORINFO` structure in bytes (on input).

`enum NVCONNECTORLAYOUT nConnectorLayout;`	Connector layout:
	`NVCONNECTORLAYOUT_UNKNOWN`
	`NVCONNECTORLAYOUT_CARD_SINGLESLOT`
	`NVCONNECTORLAYOUT_CARD_DOUBLESLOT`
	`NVCONNECTORLAYOUT_CARD_MOBILE_MXM`
	`NVCONNECTORLAYOUT_CARD_MOBILE_OEM`
	`NVCONNECTORLAYOUT_MOBILE_BACK`
	`NVCONNECTORLAYOUT_MOBILE_BACK_LEFT`
	`NVCONNECTORLAYOUT_MOBILE_BACK_DOCK`
	`NVCONNECTORLAYOUT_NFORCE_STANDARD`

`DWORD dwConnectorCount;`	Number of connectors on the graphics card.

NvEnumPhysicalConnectorDetails()

The function `NvEnumPhysicalConnectorDetails()` returns detailed information on a physical connector on the graphics card. This function reads information from an optional block of the NV4X and higher graphics card BIOS. When used with graphics cards that do not have this optional block, the function returns NV_NOTSUPPORTED.

Earliest Driver	`65.91`
Function Prototype	`NVRESULT NVAPIENTRY NvEnumPhysicalConnectorDetails ( IN UINT nAdapterNumber, IN DWORD dwConnectorIndex, OUT NVCONNECTORDETAIL* pConnectorDetail );`
Parameters In	`UINT nAdapterNumber --` Windows display number for the graphics card. `DWORD dwConnectorIndex --` Index of physical connector (range defined by NVCONNECTORINFO.dwConnectorCount)
Parameter Out	`NVCONNECTORDETAIL* pConnectorDetail` Pointer to receive detailed physical connector information. Caller should initialize pConnectorDetail->cbSize to sizeof(NVCONNECTORDETAIL).
Return Values	**`NV_OK`** : Success. **`NV_NOTSUPPORTED:`** Unsupported feature (requires compatible video BIOS). **`NV_NOMORE`** : Index of physical connector does not exist.

Table 3.3 NVCONNECTORDETAIL Content

Datas	Description
`DWORD cbSize;`	Size of the `NVCONNECTORDETAIL` structure in bytes (on input).

`enum NVCONNECTORTYPE nConnectorType;`	Connector type:
	`NVCONNECTORTYPE_UNKNOWN`
	`NVCONNECTORTYPE_UNCLASSIFIED_ANALOG`
	`NVCONNECTORTYPE_UNCLASSIFIED_DIGITAL`
	`NVCONNECTORTYPE_UNCLASSIFIED_TV`
	`NVCONNECTORTYPE_UNCLASSIFIED_LVDS`
	`NVCONNECTORTYPE_VGA`
	`NVCONNECTORTYPE_DVI_A`
	`(DVI Analog)`
	`NVCONNECTORTYPE_DVI_D`
	`(DVI Digital_`
	`NVCONNECTORTYPE_DVI_I`
	`(DVI Integrated)`
	`NVCONNECTORTYPE_DVI_I_TV_SVIDEO`
	`NVCONNECTORTYPE_DVI_I_TV_COMPOSITE`
	`NVCONNECTORTYPE_DVI_I_TV_SVIDEO_BREAKOUT_COMPOSITE`
	`NVCONNECTORTYPE_LFH_DVI_I_1`
	`( 60-pin LFH, as in Quadro NVS)`
	`NVCONNECTORTYPE_LFH_DVI_I_2`
	`( 60-pin LFH connector)`
	`NVCONNECTORTYPE_LVDS_SPWG`
	`(as in laptop panels)`
	`NVCONNECTORTYPE_LVDS_OEM`
	`NVCONNECTORTYPE_TMDS_OEM`
	`NVCONNECTORTYPE_ADC`
	`NVCONNECTORTYPE_TV_COMPOSITE`
	`NVCONNECTORTYPE_TV_SVIDEO`
	`NVCONNECTORTYPE_TV_SVIDEO_BREAKOUT_COMPOSITE`
	`NVCONNECTORTYPE_TV_SCART`
	`NVCONNECTORTYPE_PC_YPRPB`
	`( Personal Cinema - YPrPb)`
	`NVCONNECTORTYPE_PC_SVIDEO`
	`( Personal Cinema - S-Video)`
	`NVCONNECTORTYPE_PC_COMPOSITE`
	`( Personal Cinema - Composite)`
	`NVCONNECTORTYPE_STEREO`
	`( 3-Pin DIN Stereo Connector)`

Table 3.3 NVCONNECTORDETAIL Content

Datas	Description
`DWORD dwConnectorLocation;`	Connector location (for add-in cards zero means the connector furthest from the motherboard)

`DWORD dwFlags;`	Connector flags :
	`NVCONNECTORFLAG_REMOVEABLE 0x00000001`
	Connector supports removeable devices (an example of a fixed connector is an internal laptop display)
	`NVCONNECTORFLAG_DIGITAL 0x00000002`
	Connector supports digital displays (ex. DFPs)
	`NVCONNECTORFLAG_ANALOG 0x00000004`
	Connector supports analog displays (ex. CRTs)
	`NVCONNECTORFLAG_TV 0x00000008`
	Connector supports TV sets
	`NVCONNECTORFLAG_HDTV 0x00000010`
	Connector supports HDTV sets

`DWORD dwConnectorCount;`	Number of connectors on the card.

PowerMizer Functions

The PowerMizer API provides functions that read and write the PowerMizer level to be used when a laptop is running either on battery or AC power. AC and battery power each have three PowerMizer levels, described as follows:

Value	PowerMizer Level
1	Maximum performance
2	Balanced
3	Maximum power savings

nvGetPwrMzrLevel()

This function gets the current PowerMizer level for AC and battery power.

Function Prototype	`BOOL nvGetPwrMzrLevel(OUT DWORD* pdwBatteryLevel,`
	`OUT DWORD* pdwACLevel);`

Parameters Out	`DWORD*` must be a valid pointer. `pdwBatteryLevel`: Value in the range of 1–3. `pdwACLevel`: Value in the range of 1–3.
Return Values	**`True`** if the PowerMizer level is obtained successfully. **`False`** for the following reasons: • The `DWORD*` pointer is not valid. • The system does not support PowerMizer. • The value passed in is less than 1 or greater than 3. • The hardware escape into the resource manager to obtain the PowerMizer level fails.

nvSetPwrMzrLevel()

This function sets the PowerMizer level for AC and battery power.

Function Prototype

```
BOOL nvSetPwrMzrLevel(IN DWORD* pdwBatteryLevel,
                                IN DWORD* pdwACLevel);
```

Parameters In

`DWORD*` must be a valid pointer.

`pdwBatteryLevel`: Value in the range of 1–3.

`pdwACLevel`: Value in the range of 1–3.

Return Values

`True` if the PowerMizer level is obtained successfully.

`False` for the following reasons:

- The `DWORD*` pointer is not valid.
- The system does not support PowerMizer.
- The value passed in is less than 1 or greater than 3.
- The hardware escape into the resource manager to obtain the PowerMizer level fails.

Temperature and Power Monitoring

NvCplGetThermalSettings()

Function Prototype

```
BOOL CDECL NvCplGetThermalSettings
     (IN UINT nWindowsMonitorNumber,
      OUT DWORD* pdwCoreTemp,
      OUT DWORD* pdwAmbientTemp,
       OUT DWORD* pdwUpperLimit);
```

Parameters In

`UINT nWindowsMonitorNumber --` The display number shown on the Windows Display Properties->Settings page.
A value of 0 indicates the current primary Windows display device.

`DWORD*` must be a valid pointer --

`pdwCoreTemp` -- GPU temperature in degrees Celsius.

`pdwAmbientTemp` -- Ambient temperature in degrees Celsius.

`pdwUpperLimit` -- Upper limit of the GPU temperature specification.

Return Values

`True` on success.

`False` on failure.

NvCplIsExternalPowerConnectorAttached()

This API determines if there is power at the external power connector of the NVIDIA graphics card.

Earliest Driver	`61.60 (Release 60)`
	`65.11 (Release 65)`

Function Prototype	`BOOL CDECL  NvCplIsExternalPowerConnectorAttached (IN BOOL* pbAttached)`
Parameters Out	`BOOL* pbAttached --` Pointer to the flag indicating the external power status. **`FALSE`**: No power at the external connector. **`TRUE`**: There is power at the external connector.
Return Values	**`True`** on success. **`False`** on failure.

Data Control

The APIs in this section are used to get and set values for one of the settings listed in Table 3.4.

Table 3.4 NvCplGetDataInt and NvCplSetDataInt Settings and Values

Setting Index	Description / Values
`NVCPL_API_AGP_BUS_MODE`	Type of graphics card connection in the system `1` : PCI `4` : AGP `8` : PCI Express
`NVCPL_API_VIDEO_RAM_SIZE`	Graphics card video RAM in megabytes
`NVCPL_API_TX_RATE`	For AGP systems, the graphics card AGP bus rate (1x, 2x, ...) For PCI-Express systems, the PCI-Express bus width (for example, x1, x16) (***Effective in version 71.70***.)
`NVCPL_API_CURRENT_AA_VALUE`	Graphics card antialiasing setting. `0` : Off `1` : 2x `2` : 2x Quincunx `3` : 4x `4` : 4x Gaussian `5` : 4xS `6` : 6xS `7` : 8xS `8` : 16x
`NVCPL_API_AGP_LIMIT`	Graphics card GART size
`NVCPL_API_FRAME_QUEUE_LIMIT`	The maximum number of frames that can be prerendered by the driver.
`NVCPL_API_NUMBER_OF_GPUS` ***(Effective in 65.60)***	The number of enabled GPUs in the system.

Table 3.4 NvCplGetDataInt and NvCplSetDataInt Settings and Values

Setting Index	Description / Values
`NVCPL_API_NUMBER_OF_SLI_GPUS` (***Effective in 65.60***)	Graphics card number of enabled SLI GPUs in the system.
`NVCPL_API_SLI_MULTI_GPU_RENDERING_MODE` ***(Effective in 71.10)***	Get/Set SLI multi-GPU rendering mode. dwValue is a bit mask defined as follows: 0x10000000 : SLI mode enabled 0x00000000 : Autoselect the SLI rendering mode Setting more then one of the following bits has the same effect as autoselect. 0x00000001 : Cooperative rendering 0x00000002 : Proportional rendering 0x00000004 : Single GPU

NvCplGetDataInt()

Function Prototype
```
BOOL CDECL nvCplGetDataInt( IN DWORD dwSettingIndex
                            IN DWORD* pdwValue );
```

Parameters In `DWORD dwSettingIndex --`

One of the settings listed in Table 3.4 .

Parameters Out `DWORD* pdwValue` -- must point to valid storage in caller space for the corresponding data.

Return Values **`True`** on success.

`False` on failure.

NvCplSetDataInt()

Function Prototype
```
BOOL CDECL nvCplSetDataInt( IN DWORD dwSettingIndex
                            IN DWORD dwValue );
```

Parameters In `DWORD dwSettingIndex --`

`NVCPL_API_FRAME_QUEUE_LIMIT` -- 6

`DWORD dwValue` -- Value between 0 and 255.

Return Values **`True`** on success.

`False` on failure.

TV Functions

NvGetTVConnectedStatus()

This function has been deprecated, but is available for legacy support. New applications should use NvGetDisplayInfo()**.**

This API returns the TV connector type even when the TV is not enabled.

Earliest Driver	`57.60 (Release 55)`
	`60.60 (Release 60)`

Function Prototype	`BOOL CDECL NvGetTVConnectedStatus`
	`            ( OUT DWORD* pdwConnected);`

Parameters Out	`DWORD* pdwConnected` is a pointer to a bitmask that specifies the TV connector type or types --
	`NVAPI_TV_ENCODER_CONNECTOR_UNKNOWN      0x0`
	`NVAPI_TV_ENCODER_CONNECTOR_SDTV         0x1`
	`NVAPI_TV_ENCODER_CONNECTOR_HDTV         0x2`

Return Values	**`True`** on success. **`False`** on failure.

NvGetCurrentTVFormat()

This function has been deprecated, but is available for legacy support. New applications should use* NvGetDisplayInfo()*.

Earliest Driver	`53.30`

Function Prototype	`BOOL CDECL NvGetCurrentTVFormat`
	`        ( OUT DWORD* pdwFormat);`

Parameters Out	`DWORD* pdwFormat` is a pointer to the current TV format defined as follows -- `0`: NTSC_M `1`: NTSC_J `2`: PAL_M `3`: PAL_A `4`: PAL_N `5`: PAL_NC `8`: TV_STANDARD_HD576i `9`: TV_STANDARD_HD480i (D1 connector) `10`: TV_STANDARD_HD480p (D2 connector) `11`: TV_STANDARD_HD576p `12`: TV_STANDARD_HD720p (D4 connector) `13`: TV_STANDARD_HD1080i (D3 connector) `14`: TV_STANDARD_HD1080p (D5 connector) `16`: TV_STANDARD_HD720i
Return Values	**`True`** on success. **`False`** on failure.

NvSetHDAspect()

This API sets the HDTV aspect ratio

Earliest Driver	`53.30`
Function Prototype	`BOOL CDECL NvSetHDAspect( IN DWORD* pdwAspect);`
Parameters In	`DWORD* pdwAspect` is a pointer to the HDTV aspect ratio defined as follows -- `NVAPI_ASPECT_FULLSCREEN` `0` : 4:3 aspect ratio `NVAPI_ASPECT_LETTERBOX` `1` : 4:3 aspect ratio, letterbox `NVAPI_ASPECT_WIDESCREEN` `2` : 16:9 aspect ratio
Return Values	**`True`** on success. **`False`** on failure.

NvSetTVWideScreenSignalling()

This API controls the widescreen signalling.

Earliest Driver `57.20`

Function Prototype

```
LRESULT CDECL NvSetTVWideScreenSignalling
        (DWORD* pdwTVType, DWORD* pdwData );
```

Parameters In

`DWORD* pdwTVType` is a pointer to the TV format defined as follows --

```
  #define NVAPI_TV_NONE 0
  #define NVAPI_TV_NTSC 1
  #define NVAPI_TV_PAL 2
  #define NVAPI_TV_HD 3
```

`DWORD* pdwData` is a pointer to data corresponding to the TV type defined as follows --

```
  #define NVTV_WSS_NTSC_IEC61880__ASPECT 1:0
  #define NVTV_WSS_NTSC_IEC61880__WORD1 5:2
  #define NVTV_WSS_NTSC_IEC61880__WORD2 13:6
  #define NVTV_WSS_PAL_ETSI300294__ASPECT 3:0
  #define NVTV_WSS_PAL_ETSI300294__ENHANCED 7:4
  #define NVTV_WSS_PAL_ETSI300294__SUBTITLES 10:8
  #define NVTV_WSS_PAL_ETSI300294__SURROUND 11:11
```

Return Values

Error codes --

```
#define NVAPI_OPERATION_SUCCEEDED 0
#define NVAPI_ERROR_INVALID_INPUT 1
#define NVAPI_ERROR_NO_TV 2
#define NVAPI_ERROR_FAILED_INITIALIZATION 3
#define
NVAPI_ERROR_HARDWARE_DOESNT_SUPPORT_FEATURE 4
#define
NVAPI_ERROR_SETTING_INCONGRUENT_WITH_MODALITY 5
#define NVAPI_WARNING_WSS_INCONGRUENT_WITH_CP 6
#define NVAPI_ERROR_UNKNOWN 7
```

NVTVOutManageOverscanConfiguration()

This API sets up the overscan configuration—overscan, underscan, or native mode—based on the specified TV format.

Earliest Driver `61.20`

Function Prototype

```
BOOL CDECL NVTVOutManageOverscanConfiguration
          ( IN DWORD dwTVFormat,
            IN OUT DWORD* pdwOverscanFlags,
             IN BOOL bGet);
```

Parameters In

`DWORD dwTVFormat --` must be one of the following HDTV format standards:

```
TV_STANDARD_HD480i
TV_STANDARD_HD480p
TV_STANDARD_HD576p
TV_STANDARD_HD720p
TV_STANDARD_HD1080p
TV_STANDARD_HD576i
TV_STANDARD_HD1080i
```

`DWORD* pdwOverscanFlags --` pointer to the overscan configuration data to set, if this is a set call.

```
NVCPL_API_OVERSCAN_SHIFT   0x00000010
NVCPL_API_UNDERSCAN        0x00000020
NVCPL_API_NATIVEHD         0x00000080
```

`BOOL bGet --` determines whether this is a set or get call:

`TRUE`: Get (read)

`FALSE`: Set (write)

Parameters Out

`DWORD* pdwOverscanFlags --` pointer to the overscan configuration data received, if this is a get call.

```
NVCPL_API_OVERSCAN_SHIFT   0x00000010
NVCPL_API_UNDERSCAN        0x00000020
NVCPL_API_NATIVEHD         0x00000080
```

Return Values

`True` on success.

`False` on failure.

TV VBI Functions

NvTVQueryVBI()

This API queries the TV VBI data, such as copy control, wide-screen signalling, or TV captioning.

***This API is not supported in Release 75 drivers and later*.**

Earliest Driver	`61.20`
Function Prototype	`TVRESULT CDECL NvTVQueryVBI ( OUT NVAPI_TV_VBI* pVBI );`
Parameters In	`None`
Parameters Out	`NVAPI_TV_VBI* pVBI` -- Pointer to the VBI parameters. See Table 3.5, "NVAPI_TV_VBI Content for NvTVConfigureVBI() and NvTVQueryVBI()," on page 73.
Return Values	Error codes:

```
#define NVAPI_TV_VBI_SUCCESS                        0
#define NVAPI_TV_VBI_ERROR__NOT_SUPPORTED           1
#define NVAPI_TV_VBI_ERROR__BAD_PARAMETER           2
#define NVAPI_TV_VBI_ERROR__TYPES_CONFLICT          3
#define NVAPI_TV_VBI_ERROR__VIDEO_CONFLICT          4
#define NVAPI_TV_VBI_ERROR__SIZE_MISMATCH           5
#define NVAPI_TV_VBI_ERROR__VERSION_MISMATCH        6
#define NVAPI_TV_VBI_ERROR__DLL_UNINITIALIZED       7
#define NVAPI_TV_VBI_ERROR__INTERNAL_ERROR          8
#define NVAPI_TV_VBI_ERROR__TYPE_UNSUPPORTED        9
```

NvTVConfigureVBI()

This API sets up the TV VBI data, such as copy control, wide-screen signalling, or TV captioning.

***This API is not supported in Release 75 drivers and later*.**

Earliest Driver `61.20`

Function Prototype

```
TVRESULT CDECL NvTVConfigureVBI
                    ( IN NVAPI_TV_VBI* pVBI );
```

Parameters In `NVAPI_TV_VBI* pVBI` -- Pointer to the VBI parameters. See Table 3.5, “NVAPI_TV_VBI Content for NvTVConfigureVBI() and NvTVQueryVBI(),” on page 73.

Parameters Out `None`

Return Values Error codes:

```
#define NVAPI_TV_VBI_SUCCESS                    0
#define NVAPI_TV_VBI_ERROR__NOT_SUPPORTED       1
#define NVAPI_TV_VBI_ERROR__BAD_PARAMETER       2
#define NVAPI_TV_VBI_ERROR__TYPES_CONFLICT      3
#define NVAPI_TV_VBI_ERROR__VIDEO_CONFLICT      4
#define NVAPI_TV_VBI_ERROR__SIZE_MISMATCH       5
#define NVAPI_TV_VBI_ERROR__VERSION_MISMATCH    6
#define NVAPI_TV_VBI_ERROR__DLL_UNINITIALIZED   7
#define NVAPI_TV_VBI_ERROR__INTERNAL_ERROR      8
#define NVAPI_TV_VBI_ERROR__TYPE_UNSUPPORTED    9
```

Table 3.5 NVAPI_TV_VBI Content for NvTVConfigureVBI() and NvTVQueryVBI()

Data Field	Description
`DWORD size`	Size of the `NVAPI_TV_VBI` structure (in bytes); set this field to `sizeof(NVAPI_TV_VBI)`.
`DWORD version`	Set to `NVAPI_TV_VBI_APIVERSION` to indicate the version level of the structure.

Table 3.5 NVAPI_TV_VBI Content for NvTVConfigureVBI() and NvTVQueryVBI()

Data Field	Description

`DWORD type`	Type of VBI data.
	For NvTVQueryVBI():
	- If the caller sets 'type' = 0, t
	the bitmask of the supported VBI types are returned.
	- If the caller sets 'type' = bitmask,
	information about each requested VBI type is returned.
	For NvTVConfigureVBI():
	- Caller must set 'type' to one or more of the following:
	`NVAPI_TV_VBI_TYPE_NONE 0x00000000;`
	`NVAPI_TV_VBI_TYPE_IEC61880 0x00000001;`
	`NVAPI_TV_VBI_TYPE_ETSI_EN300294 0x00000002;`
	`NVAPI_TV_VBI_TYPE_CEA805A_TYPEA 0x00000004;`
	`NVAPI_TV_VBI_TYPE_CEA805A_TYPEB 0x00000008;`
	`NVAPI_TV_VBI_TYPE_MACROVISION 0x00000010;`
	`NVAPI_TV_VBI_TYPE_EIA608B 0x00000020;`
	`NVAPI_TV_VBI_TYPE_EIAJ_CPR1204 0x00000040;`

`DWORD reserved0;`	
`DWORD reserved1;`	

`TV_VBI_EIA608B eia608B;`	EIA 608B captioning information.
	`DWORD on;`
	`DWORD count;`
	`WORD oddfield_16[30];`
	`WORD evenfield_16[30];`
	Each odd field is transmitted before each even field.

Table 3.5 NVAPI_TV_VBI Content for NvTVConfigureVBI() and NvTVQueryVBI()

Data Field	Description

`TV_VBI_EIAJCPR1204 eiajcpr1204;`	EIAJCPR1204 copy control system information. Also used for EIAJCPR1204-1 and EIAJCPR1204-2
	`DWORD on;`
	`DWORD aspect_2;`
	`0x00000000` -- 4x3 Normal
	`0x00000001` -- 16x9 Anamorphic
	`0x00000002` -- 4x3 Letterbox
	`DWORD word1_4;`
	`0x00000000` -- CGMS-A
	`0x00000008` -- Record Date
	`0x00000004` -- Record time
	`0x0000000C` -- Time remaining
	`0x00000002` -- 3D
	`0x0000000A` -- Source information
	`0x00000006` -- Signal format
	`0x0000000E` -- Package ID
	`0x00000001` -- Category
	`0x00000009` -- Control
	`0x00000005` -- Character
	`0x0000000F` -- No information
	`DWORD word2_8;`
	`0x00000000` -- CGMSA. Copy always
	`0x00000001` -- CGMSA. Copy once
	`0x00000003` -- CGMSA. Copy never

Table 3.5 NVAPI_TV_VBI Content for NvTVConfigureVBI() and NvTVQueryVBI()

Data Field	Description

`TV_VBI_ETSIEN300294`
`etsien300294`
	WTSI EN 300 294 -- 625-line wide screen signalling (WSS) information.
	`DWORD on;`
	`DWORD aspect_4;`
	`0x8` -- 4:3 full screen aspect ratio
	`0x1` -- 14:9 letterbox centered aspect ratio
	`0x2` -- 14:9 letterbox top
	`0xB` -- 16:9 letterbox centered
	`0x4` -- 16:9 letterbox top
	`0xD` -- >16:9 letterbox centered
	`0xE` -- 4:3 displayed as 14:9 letterbox centered
	`0x7` -- 16:9 anamorphic widescreen aspect ratio
	`DWORD mode_1;`
	`0x0` -- Camera (default)
	`0x1` -- Film. Recommended for still picture transmissions
	`DWORD colour_1;`
	`0x0` -- Standard (default)
	`0x1` -- "Motion Adaptive Color Plus" except when in film mode.
	`DWORD helper_1;`
	`0x0` -- None (default)
	`0x1` -- Modulated. Helper signal present (see notes in ETSI EN 300 294)
	`DWORD reserved_1;`
	`DWORD teletext_1;`
	`0x0` -- No teletext subtitles
	`0x1` -- Teletext subtitles
	`DWORD subtitles_2;`
	`0x0` -- No subtitles
	`0x1` -- Inside active
	`0x2` -- Outside active
	`DWORD surround_1;`
	`0x0` -- No surround
	`0x1` -- Surround
	`DWORD copyright_1;`
	`0x0` -- No copyright
	`0x1` -- Copyright asserted
	`DWORD generation_1;`
	`0x0` -- Generation allowed
	`0x1` -- Generation restricted

Table 3.5 NVAPI_TV_VBI Content for NvTVConfigureVBI() and NvTVQueryVBI()

Data Field	Description

`TV_VBI_IEC61880 iec61880;`	IEC 61880 copy control system information
	Also use for IEC61880-2
	`DWORD on;`
	`DWORD aspect_2;`
	`0x00000000` -- 4x3 Normal
	`0x00000001` -- 16x9 Anamorphic
	`0x00000002` -- 4x3 Letterbox
	`DWORD word1_4;`
	`0x00000000` -- CGMS-A
	`0x0000000F` -- No info
	`DWORD word2_8;`
	`0x00000000` -- CGMS-A. Copy always, PSP off, No analogue
	`0x00000001` -- CGMS-A. Copy once
	`0x00000003` -- CGMS-A. Copy never
	`0x00000008` -- CGMS-A. PSP on
	`0x00000004` -- CGMS-A. PSP 2 Burst
	`0x0000000C` -- CGMS-A. PSP 4 Burst
	`0x00000010` -- CGMS-A. Analogue

Table 3.5 NVAPI_TV_VBI Content for NvTVConfigureVBI() and NvTVQueryVBI()

Data Field	Description

`TV_VBI_CEA805A_TYPEA cea805A_typeA;`	CEA 805A Type A copy control system information
	`DWORD on;`
	`DWORD cgmsa_2;`
	`0x00000000` -- CGMS-A. Copy always
	`0x00000002` -- CGMS-A. Copy no more
	`0x00000001` -- CGMS-A. Copy once
	`0x00000003` -- CGMS-A. Copy never
	`DWORD apstrigger_2;`
	`0x00000000` -- PSP off
	`0x00000008` -- PSP one
	`0x00000004` -- PSP 2 Burst
	`0x0000000C` -- PSP 4 Burst
	`DWORD analogue_1;`
	`0x00` -- No analog
	`0x10` -- Analog

`TV_VBI_CEA805A_TYPEB cea805A_typeB;`	CEA 805A Type B copy control system information
	`DWORD on;`
	`DWORD aspect_2;`
	`DWORD analogue_1;`
	`DWORD afdvalid_1;`
	`DWORD barvalid_1;`
	`DWORD scaninfo_2;`
	`DWORD colorimetry_4;`
	`DWORD activeformat_4;`
	`DWORD redistcontrol_1;`
	`DWORD copyright_1;`
	`DWORD generation_1;`
	`DWORD apstrigger_2;`
	`BYTE  bardata[8];`

Result Codes

NvGetLastError()

For NVIDIA API function calls that return an NVRESULT error type, this function returns the result code for the last function call made in the current process.

Earliest Driver	62.90 (Rel60) / 65.40 (Rel65)
Function Prototype	`NVRESULT NVAPIENTRY NvGetLastError()`
Parameters In	N/A
Parameters Out	N/A
Returns	**`NVRESULT`** -- Result code from last API function call

NvGetLastErrorMessage()

For NVIDIA API function calls that return an NVRESULT error type, this function returns the localized result message text for the last function call made in the current process.

The result message retrieved from this function is usually more descriptive than that returned from NvGetErrorMessage(). For example:

`NvGetErrorMessage(NV_BADPARAMETER)` returns

```
"Invalid parameter."
```

`NvGetLastErrorMessage()` might return

```
"Invalid parameter pConfig->signalFormat in function
NvGvoConfigSet()."
```

Earliest Driver	62.90 (Rel60) / 65.40 (Rel65)
Unicode Function Prototype	`LPCWSTR NVAPIENTRY NvGetLastErrorMessageW()`
ASCII Function Prototype	`LPCSTR NVAPIENTRY NvGetLastErrorMessageA()`
Parameters In	N/A
Parameters Out	N/A
Returns	**`LPC[W]STR`** - Temporary pointer to the message text for the result code.

NvGetErrorMessage()

This function returns the localized message text for the specified result code. The result message retrieved from this function is usually less description than that returned from `NvGetLastErrorMessage()`.

Earliest Driver	62.90 (Rel60) / 65.40 (Rel65)
Unicode Function Prototype	`LPCWSTR NVAPIENTRY NvGetErrorMessageW(NVRESULT nr)`
ASCII Function Prototype	`LPCSTR NVAPIENTRY NvGetErrorMessageA(NVRESULT nr)`
Parameters In	`NVRESULT nr` -- The result code, enumerated as follows:: `NV_OK` = `0`, // Success. `NV_INTERNALERROR` = `1`, // Internal error. `NV_ALREADYINITIALIZED` = `2`, // Already initialized. `NV_NOTINITIALIZED` = `3`, // Not initialized. `NV_OUTOFMEMORY` = `4`, // Not enough memory for operation. `NV_NOTSUPPORTED` = `5`, // Feature not supported. `NV_NOTAVAILABLE` = `6`, // Feature not presently available. `NV_NOTIMPLEMENTED` = `7`, // Feature not implemented. `NV_BADPARAMETER` = `8`, // Invalid parameter. `NV_ACCESSDENIED` = `9`, // Access denied. `NV_RUNNING` = `10`, // Operation requires inactive environment. `NV_NOTRUNNING` = `11`, // Operation requires active environment. `NV_FILENOTFOUND` = `12`, // Unable to locate file. `NV_NOMORE` = `13`, // No more items. `NV_ILLEGALSTATE` = `14`, // Illegal state could not be resolved. `NV_NOTFOUND` = `15`, // Not found `NV_WARN_INTERNALERROR` = `-1`, // Internal warning. `NV_WARN_ILLEGALSTATE` = `-14`, // Illegal state was automatically resolved. `NV_WARN_NOTEQUAL` = `-15`, // State compare failed `NV_WARN_NOMORE` = `-16`, // Warning that state compare failed and there are no more to enum
Parameters Out	NONE
Returns	**`LPC[W]STR`** - Temporary pointer to the message text for the result code.

Macros

Macro Define	Description
`NVRESULT_SUCCESS(nr)  ((int)(nr) <= 0)`	Success (NV_OK or NV_WARN_xxx)
`NVRESULT_FAILURE(nr)  ((int)(nr) > 0)`	Failure
`NVRESULT_WARN(nr)     ((int)(nr )< 0)`	Warning
`NVRESULT_ERRORCODE(nr) ((NVRESULT)(abs((int)nr)))`	Extract error code from NVRESULT value (warning --> error)

4 IDISPATCH COM INTERFACE

Control of several TV-out settings is supported though the COM IDispatch interface **`NvcplLateBound`**. This interface is a fully custom implementation of the IDispatch interface. This chapter assumes a familiarity with the mechanics of COM and IDispatch, and their invocation and use in C++.

`NvcplLateBound` is intended to be extensible, with no strictly defined parameters as used in other APIs documented in this manual. Instead, the programmer is directed to the following sections as guidelines to use in understanding the interface and developing an appropriate implementation:

- "Supported Function Calls" on page 84

  Describes the controls that are exposed through the interface.

- "Using NvcplLateBound" on page 88

  Provides a sample visual basic script that you can use to test the interface and familiarize yourself with the functions. The section also provides tips for implementing using C++, and includes a sample application that demonstrates use of the TV control methods.

Supported Function Calls

The function calls are invoked as strings which can be used directly from VBScript or Javascript. C++ requires a little more effort, as explained later in this chapter.

- "Calls Corresponding to Panel APIs" on page 84
- "Specifying Display Adapters" on page 85
- "TV Controls" on page 85

Calls Corresponding to Panel APIs

Table 4.1 lists the function calls–corresponding to the NVPanel API calls—that the **`NvCplLateBound`** interface supports.

Table 4.1 Miscellaneous NVPanel Calls

API Call	String
`dtcgfex()`	`"dtcgex <dtcfg commands and args>"` See "dtcfgex()" on page 24.
`NvCplGetConnectedDevicesString()`	`"NvCplGetConnectedDevicesString"` See "NvCplGetConnectedDevicesString()" on page 57.
`NvGetCurrentTVFormat()`	`"NvGetCurrentTVFormat"` See "NvGetCurrentTVFormat()" on page 69.
`NvGetDisplayInfo()`	`"NvGetDisplayInfo <device moniker> <Flag>"` See "NvGetDisplayInfo()" on page 30.
`NvGetTVConnectedStatus()`	`"NvGetTVConnectedStatus"` See "NvGetTVConnectedStatus()" on page 68.
`NvSetHDAspect()`	`"NvSetHDAspect"` See "NvSetHDAspect()" on page 69.
`NVTVOutManageOverscanConfiguration()`	`"NVTVOutManageOverscanConfiguratio"` See "NVTVOutManageOverscanConfiguration()" on page 71.

Specifying Display Adapters

The **`NvCplLateBound`** interface includes a method for selecting a particular display adapter in a multiple adapter system. This is accomplished by specifying one of the display numbers from the Windows Settings page. All subsequent function calls act upon the displays connected to that adapter.

Table 4.2 Select a Display Device

Operation	String
Select the display device.	"`SelectDisplayDevice(MonitorNumber)`" where "`MonitorNumber`" is the monitor icon number shown on the Windows Display Properties Settings page. A value of 0 selects the default display device.

TV Controls

The **`NvCplLateBound`** interface includes the following TV controls:

Flicker filter

Get and set functions are supported.

Table 4.3 Flicker Filter Control Strings

Operation	String
Get the current flicker filter value.	"`Gettvsettings flicker current`"
Get the supported flicker filter range	"`Gettvsettings flicker range`"
Get the default flicker filter value.	"`Gettvsettings flicker default`"
Set the flicker filter value.	"`Settvsettings flicker current <value>`"

Overscan

Get and set functions are supported.

Table 4.4 TV Overscan Control Strings

Operation	String
Get the current TV overscan value.	"`Gettvsettings overscan current`"
Get the supported TV overscan range.	"`Gettvsettings overscan range`"
Get the default TV overscan value.	"`Gettvsettings overscan default`"
Set the TV overscan value.	"`Settvsettings overscan current <value>`"

Position Control

Set functions are supported.

Table 4.5 Position Control Strings

Operation	String
Move the TV position up.	`“Settvsettings position up <value>”` where `<value>` must be positive
Move the TV position down.	`“Settvsettings position down <value>”` where `<value>` must be negative
Move the TV position to the right.	`“Settvsettings position right <value>”` where `<value>` must be positive
Move the TV position to the left.	`“Settvsettings position left <value>”` where `<value>` must be negative
Restore TV screen to the default position.	`“Settvsettings position restore”`

Saturation

Get and set functions are supported.

Table 4.6 Saturation Control Strings

Operation	String
Get the current TV saturation value.	`“Settvsettings saturation current”`
Get the supported TV saturation range.	`“Settvsettings saturation range”`
Get the default TV saturation.	`“Settvsettings saturation default”`
Set the specified TV saturation value.	`“Settvsettings saturation current <value>”`

Setting All Controls to Default Values

Refer to the individual controls to set a particular default value. To set all of the controls (flicker filter, overscan, saturation, and position) the default values, use the following command:

```
“Settvsettings all default”
```

Tips on Setting TV Values

Because the available TV settings and range depend on the hardware, such as the GPU and TV encoder, NVIDIA recommends obtaining the range and current value of a TV setting before attempting to set a value.

For example, to set the flicker filter:

1. Get the current flicker filter value ("Gettvsettings flicker current")
2. Get the flicker filter range ("Gettvsettings flicker range")

   If a value of "0" is returned for both queries, then flicker filtering is not supported by the hardware and no attempt should be made to set a value.
3. If supported, set a flicker filter value that is within the range. ("Settvsettings flicker current <value>")

Using NvcplLateBound

Sample Test Script

This VB script tests the interface and allows it to be exercised via script/ dispatch interface. It provides an overview of how to use **`nvcpllatebound`**. This example uses mixed case parameter strings to show that they are not case sensitive.

The script should be invoked from either an HTML page in Internet Explorer, or as "`wscript testnvcplscript.vbs`".

```
**************************************************************************************
rem MsgBox "starting Note that you need to regsvr32 nvcpl.dll from c:\windows\system32
if you did not use the driver install package***
set r = createobject("nvcpl.nvcpllatebound") ' Create the nvcpl com object
msgbox r.geTTvsettings("current", "Flicker")
msgbox r.gettvSeTtings("range", "flicker")
msgbox r.gettvSeTtings("range", "Overscan")
msgbox r.gettvSeTtings("cuRRENT", "OvErscan")
msgbox "Making set calls"
r.settvSeTtings 2, "current", "flicker"
msgbox r.geTTvsettings("current", "Flicker")
r.settvsettings r.geTTvsettings("current", "Flicker") + 1 , "current", "flicker"
msgbox r.geTTvsettings("current", "Flicker")
r.settvsettings 22, "up","position"
Msgbox "Calling Position-restore"
r.settvsettings 0,"restore", "position"
**************************************************************************************
```

Using C++

This section explains how to use the interface using C++. See section "Code Example: NvcplDispinterface" on page 89 for a full code example.

Instantiation

```
DEFINE_GUID(CLSID_CplLateBound, 0x11556518, 0xf20d, 0x49ec, 0xa5, 0x31,
0xe0, 0xbd, 0xdd, 0x5e, 0x66, 0x60);
```

Alternatively, use the `ProgID` (preferred) as "`NVCpl.NvCplLateBound`" for a more decoupled interface.

Code Example: NvcplDispinterface

This code sample demonstrates the manner and arguments for invoking the methods `Gettvsettings` and `Settvsettings` using the IDispatch compatible interface. This example uses mixed case parameter strings to show that they are not case sensitive.

```
*******************************************************************************************
// NvcplDispinterface.cpp : Defines the entry point for the console application.
// An example to illustrate how to work the TV Settings


#include <afxwin.h>
#include "stdafx.h"
//
//#define _WIN32_WINNT 0x0400
#include <atlbase.h>
#define INITGUID
#include <initguid.h>


DEFINE_GUID( CLSID_NvCplLateBound , 0x11556518, 0xF20D, 0x49EC, 0xA5, 0x31, 0xE0, 0xBD,
0xDD, 0x5E, 0x66, 0x60 );


CComModule _Module;


#include <atlcom.h>


HRESULT NVCPL_InvokeWrapper2StringArgs( IDispatch *p, TCHAR *szFuncName, TCHAR *argv[],
VARIANT *pVtResult );


HRESULT NVCPL_InvokeWrapper2String1NumberArgs( IDispatch *p , TCHAR *szFuncName,
                                      TCHAR *argv[], int Number, VARIANT  *pVtResult );


int _tmain(int argc, _TCHAR* argv[])
{
    printf( "Demonstrating The NVCPL IDispatch Interface calls ...\n" );


    CoInitialize( NULL );


    try
    {
        CComPtr<IDispatch> spDisp;
        HRESULT hr = ::CoCreateInstance( CLSID_NvCplLateBound, NULL,
                                         CLSCTX_INPROC_SERVER, IID_IDispatch,
(LPVOID*)&spDisp );

        if  ( (spDisp == NULL) || (hr != S_OK) )
        {
            printf("Cannot create NVIDIA Display Panel COM server instance.\n");
            return 1 ;
        }

       // ------------------------------------------
       // GetTvSettings Flicker Range &vtOut.intVal
       // ------------------------------------------
       CComVariant vtResult;

       char *argParam[] = {"Flicker","Range"};
       hr = NVCPL_InvokeWrapper2StringArgs(spDisp.p, "GetTvSettings", argParam ,
                                                    &vtResult);
       printf( "Result of GetTvSettings Flicker Range => %d\n" , vtResult.intVal );

       int range_flicker = vtResult.intVal ;

       // ------------------------------------------
       // GetTvSettings Flicker current &vtOut.intVal
       // ------------------------------------------
       vtResult.Clear();
       char *argParam1[] = {"Flicker","cuRRent"};
       hr = NVCPL_InvokeWrapper2StringArgs(spDisp.p, "GetTvSettings", argParam1 ,
                                                     &vtResult);
       printf( "Result of GetTvSettings Flicker current => %d\n" , vtResult.intVal );
       int current_flicker = vtResult.intVal;
```

```
// -------------------------------------------------------------------
// -----------  SET TV Flicker Current (Value)
// -------------------------------------------------------------------

// first, we need to ensure that flicker is supported on this tv encoder (a good
// visual check is to see if the Control panel tvout settings have it too)

if  ( range_flicker <= 0 )
{
    // Handle the error!!
}
else
{
    // Note: You cannot set the range, that is read only!!
    // you can however set the current value
    // set the current flicker to some value that is less than the range
    int value_flicker_to_set = current_flicker + ( range_flicker -
                                                        current_flicker ) / 2 ;
   // but this could be any +ve value less than the range

    vtResult.Clear();
    char *argParam6[] = {"fliCker","Current"};
    hr = NVCPL_InvokeWrapper2String1NumberArgs( spDisp.p, "SetTvSettings",
                               argParam6, value_flicker_to_set, &vtResult );

    // Read the value we just set
    hr = NVCPL_InvokeWrapper2StringArgs( spDisp.p, "GetTvSettings", argParam6 ,
                                                &vtResult );
  printf( "Result of SEtTvSettings flicker current => %d\n" , vtResult.intVal );
}

// -------------------------------------------
// GetTvSettings overscan Range &vtOut.intVal
// -------------------------------------------
vtResult.Clear();
char *argParam9[] = { "overscan","Range" };
hr = NVCPL_InvokeWrapper2StringArgs( spDisp.p, "GetTvSettings", argParam9 ,
                                         &vtResult );
printf( "Result of GetTvSettings overscan Range => %d\n" , vtResult.intVal );
int range_overscan = vtResult.intVal;
```

```
// ------------------------------------------
// GetTvSettings overscan current &vtOut.intVal
// ------------------------------------------
vtResult.Clear();
char *argParam10[] = {"OverScan","cuRRent"};
hr = NVCPL_InvokeWrapper2StringArgs( spDisp.p, "GetTvSettings", argParam10 ,
                                          &vtResult );
printf( "Result of GetTvSettings overscan current => %d\n" , vtResult.intVal );
int current_overscan = vtResult.intVal;

// -----------------------------------------------------------------
// -----------  SET TV Overscan Current (Value)
// -----------------------------------------------------------------

if  ( range_overscan <= 0 )
{
    // Handle the error!!
}
else
{
    // Note: You cannot set the range, that is read only!!
    // you can however set the current value
    // set the current overscan to some value that is less than the range
    int value_overscan_to_set = current_overscan + ( range_overscan -
                                                    current_overscan ) / 2 ;

    vtResult.Clear();
    char *argParam14[] = {"OverscAn","Current"};

    hr = NVCPL_InvokeWrapper2String1NumberArgs( spDisp.p, "SetTvSettings",
                             argParam14, value_overscan_to_set, &vtResult );

    // Read the value we just set
    hr = NVCPL_InvokeWrapper2StringArgs( spDisp.p, "GetTvSettings", argParam14 ,
                                              &vtResult );
 printf( "Result of SEtTvSettings overscan current => %d\n" , vtResult.intVal );
}
```

```
        // ------------------------------------------------------------------
        // -----------  SET TV Position UP (Value)
        // ------------------------------------------------------------------

        vtResult.Clear();
        char *argParam15[] = {"Position","UP"};
        hr = NVCPL_InvokeWrapper2String1NumberArgs( spDisp.p, "SetTvSettings",
argParam15, 12, &vtResult );

        printf( "CALLING SEtTvSettings Position UP %d\n" , 12 );

        // ------------------------------------------------------------------
        // -----------  SET TV Position RIGHT (Value)
        // ------------------------------------------------------------------

        vtResult.Clear();
        char *argParam16[] = {"Position","right"};
        hr = NVCPL_InvokeWrapper2String1NumberArgs( spDisp.p, "SetTvSettings",
                                                         argParam16, 21, &vtResult );

        printf( "CALLING SEtTvSettings Position RIght %d\n" , 21 );

        // ------------------------------------------------------------------
        // -----------  SET TV Position restore
        // ------------------------------------------------------------------
        vtResult.Clear();
        char *argParam17[] = {"Position","Restore"};
        hr = NVCPL_InvokeWrapper2String1NumberArgs( spDisp.p, "SetTvSettings",
                                                          argParam17, 0, &vtResult );

        spDisp.Release();
        spDisp = NULL ;
    }
    catch(...)
    {}

    CoUninitialize();

    return 0 ;
}
```

```
HRESULT NVCPL_InvokeWrapper2StringArgs(IDispatch *p, TCHAR *szFuncName, TCHAR *argv[],
                                           VARIANT *pVtResult)
{
    if  ( !p || !szFuncName || !pVtResult )
    {
        return E_POINTER;
    }

    CComDispatchDriver dspDriver(p);
    // Get The ID of the desired Name
    USES_CONVERSION;
    DISPID  dispID = NULL;
    HRESULT hr = dspDriver.GetIDOfName(T2OLE(szFuncName),&dispID);

     if  ( hr != S_OK )
     {
         printf( "Set TV Settings : SetTvSettings, could not obtain ID of Name\n" );
         return  hr ;
     }

     // Prepare the arguments
     CComVariant varArgs[2] ;

     varArgs[0].vt = VT_BSTR;
     varArgs[0].bstrVal = (BSTR)( A2WBSTR( argv[0] ) ) ;

     varArgs[1].vt = VT_BSTR;
     varArgs[1].bstrVal = (BSTR)( A2WBSTR( argv[1] ) );

     VariantInit(pVtResult);
     pVtResult->vt = VT_EMPTY;

     CComVariant vtTmp[1];
     hr = dspDriver.InvokeN( dispID, &varArgs[0], 2, &vtTmp[0] );

     if  ( hr != S_OK )
     {
         printf("NVCPL_InvokeWrapper2StringArgs:Failed HR: 0x%08X\n",hr);
        return hr;
    }

    dspDriver.Release();

    DISPPARAMS dspParams = { vtTmp, 0, 1, 0 };
    hr = DispGetParam( &dspParams, 0, VT_I4, pVtResult, 0 );

    return  hr ;
}

HRESULT  NVCPL_InvokeWrapper2String1NumberArgs( IDispatch *p , TCHAR *szFuncName,
                                                TCHAR *argv[], int Number, VARIANT
*pVtResult )
{
    if  ( !p || !szFuncName || !pVtResult )
    {
        return E_POINTER;
    }

    CComDispatchDriver dspDriver(p);
    // Get The ID of the desired Name
    USES_CONVERSION;
    DISPID dispID = NULL;

    HRESULT hr = dspDriver.GetIDOfName(T2OLE(szFuncName),&dispID);

    if  ( hr != S_OK )
    {
        printf( "Set TV Settings : SetTvSettings, could not obtain ID of Name\n" );
        return  hr ;
    }

    // Prepare the arguments

    CComVariant varArgs[3] ;

    varArgs[0].vt = VT_BSTR;
    varArgs[0].bstrVal = (BSTR)( A2WBSTR( argv[0] ) ) ;
    varArgs[1].vt = VT_BSTR;
    varArgs[1].bstrVal = (BSTR)( A2WBSTR( argv[1] ) ) ;

    varArgs[2].vt = VT_I4;
    varArgs[2].lVal = Number;

    //pVarParams = varArgs;

    VariantInit(pVtResult);
    pVtResult->vt = VT_EMPTY;

    TCHAR  sz[ 128 ];
    ZeroMemory( sz, 128 );
    wcstombs( sz, (BSTR)CComBSTR(argv[0]), wcslen((BSTR)CComBSTR(argv[0]) ) );
    printf("first = %s (argv[0] = %s)\n",sz,argv[0]);
    ZeroMemory( sz, 128 );
    wcstombs( sz, (BSTR)CComBSTR(argv[1]), wcslen((BSTR)CComBSTR(argv[1] ) ) );
     //wcstombs( sz,(BSTR)varArgs[1], wcslen((BSTR)varArgs[1]) );
    printf("second = %s (argv[1] = %s)\n",sz,argv[1]);

    CComVariant vtTmp[1];
    hr = dspDriver.InvokeN( dispID, &varArgs[0] , 3, &vtTmp[0]);

    if  ( hr != S_OK )
    {
        printf("NVCPL_InvokeWrapper2StringArgs:Failed HR: 0x%08X\n",hr);
        return hr;
    }

    dspDriver.Release();

    DISPPARAMS dspParams = {vtTmp, 0, 1, 0 };
    hr = DispGetParam( &dspParams, 0, VT_I4, pVtResult, 0 );

    return hr;
}
```

5

NVCPL API Device Moniker Specification Version 2

This chapter describes the new device moniker scheme and explains the reasons for changing to the new scheme.

It contains the following sections:

The Need for Device Moniker Version 2

Summary of the Old Device Moniker Scheme

The previous device moniker scheme was a two-character string where the first character indicated the connection type (analog, digital, or TV) and the second character indexed each connection of that connection type.

The device moniker index was based on which bit was set in the source device mask. For example, the first moniker index assignment ("AA", "DA", or "TA") was consistently based on the first bit set.

Problem with the Old Device Moniker Scheme

There are two possible source device masks—a connected device mask, corresponding to physical connections, and an active device mask, corresponding to enabled devices (whether or not they are connected). The problem arises because different API functions use different source device masks, resulting in inconsistent meanings for the device monikers.

Resolution—Device Moniker Specification 2

The new device moniker specification includes a method for identifying whether the moniker is based on connected devices or active devices. This scheme continues to fill the needs of customers and also maintains the following functionality:

- Ability of "`dtcfg setview`" to use connected devices to attach displays that are not active.
- Ability of customers to use APIs such as `NvGetDisplayInfo`() to access displays that are physically connected, but not enabled.
- Ability of "`NvGetDisplayInfo(AA)`" to use active devices so that it can succeed even after a device is hot-plugged or unplugged.

Device Moniker Version 2 String Format

The new device moniker string format is as follows:

```
[-][#]{A,D,T}{A-H,0-7}
```

where:

- **'-' (minus)** is an optional prefix indicating that a display output is active, but no display is physically connected.

  NvCplGetActiveDevicesStrings() may return this prefix to communicate information to the application, but other API functions ignore this prefix as an input.

- **'#' (sharp)** is a prefix indicating that the device moniker is based on the connected devices mask.

  Example: "#**AA**" is the first connected device.

  The absence of the '#' prefix means that device moniker is based on the active devices mask.[1]

  Example: "**AA**" is the first active device, but might refer to the second ("#**AB**") or third ("#**AC**") physically connected device.

- **A, D, T**: indicates the type of connection driving the display:

  A = Analog, **D** = Digital, **T** = TV.

  This is *not* necessarily the display type. For example, an analog connection can drive some digital displays, or analog flat panels. Likewise, a digital connection can drive a digital CRT.

- **A–H** or **0–7:** the device moniker index.

  The device moniker index is based on which bit is set in the source device mask, where **A** (or **0**) maps to the first bit set, **B** (or **1**) maps to the second bit set, and so on.

  For example, when two displays are connected in analog mode:

  - "#AA" is the first physical display and "#AB" is the second physical display.
  - Alternatively, "#A0" can be used to reference the first physical display and "#A1" can be used to reference the second display.

1. "dtcfg setview" interprets all device monikers based on the connected devices mask. "dtcfg setview 1 standard AA" is always equivalent to "dtcfg setview 1 standard #AA".

Using Device Moniker Version 2

Using Device Monikers in NVCPL API Functions

Passing Device Monikers to the API

Passing Device Monkers with '#'

If the caller passes a device moniker prefixed with a '#', the API function will use the connected devices as the source mask.

Passing Device Monikers without '#'

If the caller passes a device moniker without this prefix, the API function uses the same source mask that it originally used:

- `dtcfg setview` uses the *connected* devices mask
- All other APIs use the *active* devices mask.

Passing Device Monikers in `pszUserDisplay`

For most NVCPL API functions, the fully-qualified device moniker is passed in the *`pszUserDisplay`* parameter as follows.

`Syntax`

[*display#*] [*devicemoniker0*] [*devicemoniker1*]

Where

`display#` is the Windows monitor number (1 to *n*) associated with the NVIDIA GPU.

If the system is set up such that a different monitor number is assigned to each display on a particular NVIDIA GPU, then any of those monitor numbers can be used to indicate that GPU.

`devicemoniker0` is the device moniker for the first display head of the GPU indicated by *`display#`*

`devicemoniker1` is the device moniker for the second display head of the GPU indicated by *`display#`*

NVIDIA recommends that the fully-qualified <display#> <devicemoniker0> <devicemoniker1> syntax be used where supported by the API functions.

Device Monikers Returned by the API

- **`NvCplGetRealConnectedDevicesString()`**

  This function returns device monikers based on the connected devices mask. This preserves existing functionality, but means device monikers will be prefixed with '#'.

- **`NvCplGetConnectedDevicesString()`**

  This function behaves as before to preserve compatibility with third-party applications that assume device monikers are two characters in length, but is not a reliable solution. *Newer applications should use NvCplGetRealConnectedDevicesString().*

- **`NvCplGetActiveDevicesString()`**

  This function returns device monikers based on the active devices mask.

  This means that device monikers will *not* have the '#' prefix, and display outputs that are active but with no display physically connected will have the '-'prefix.

- **`NvGetDisplayInfo()`**

  Two new fields have been introduced to the `NvGetDisplayInfo()` structure to convert any passed device moniker to another scheme:

  - `szConnectedMoniker` – returns monikers from connected devices, uses '#' prefix
  - `szActiveMoniker` – returns monikers from active devices, uses '-' prefix if the device is active but not connected

Examples of Device Monikers

The following table shows the device monikers for a system that has three analog displays connected but only two of them enabled, or “attached”.

Source Device Mask	CRT1 (Enabled)	CRT2	CRT3 (Enabled)
Connected	#AA	#AB	#AC
Active	AA	--	AB

The following table shows device monikers for the same system where CRT3 has been disconnected after it was enabled.

Source Device Mask	CRT1 (Enabled)	CRT2	CRT3 (Enabled, not connected)
Connected	#AA	#AB	--
Active	AA	--	-AB

Sample Output from NvGetDisplayInfo.exe

The following program output is from a system with two analog outputs driven but both are unplugged:

```
C:\NvCplAPI\NvGetDisplayInfo\Release>NvGetDisplayInfo.exe enum
Enumerating available displays...

 Display 1 [\\.\DISPLAY1 - NVIDIA GeForce FX 5900]

  Relative to connected devices,
    where "#AA" is first connected device.
    * Connected Devices          : ""
    * Connected Devices (Active) : ""

  Relative to active devices,
    where "AA" is first active device
    and "-AA" means no connection.
    * Active Devices             : "-AA,-AB"

 Output from NvCplGetMSOrdinalDeviceString,
    where the first listed device is primary
    and the later is secondary if shown.
    * Primary+Secondary Devices  : "-AB,-AA"

  Use NvGetDisplayInfo() to translate monikers that are relative
  to connected devices to be relative to active devices and vice versa.

   Query first analog output driven but unplugged:

C:\ NvCplAPI\NvGetDisplayInfo\Release>NvGetDisplayInfo.exe aa
Evaluating specified display...

 Display Number      : 1
 Connected Moniker   : Not Connected
 Active Moniker      : -AA

 Display Adapter     : "GeForce FX 5900"
 Display Board       : GeForce
 Display Device      : "\\.\DISPLAY1"
 Display Driver      : "6.14.10.7007"
 Display Mode        : Clone
 Display Qualifier   : 1b
 Display Head        : 0
 Display Type        : Cathode Ray Tube (CRT)
 Display Size        : 3200 x 2400mm (approximately 15.7")
 Display Transfer    : 2.20
 Display Optimal     : 1024 x 768 x 85 Hz
 Display Largest Safe: 1024 x 768 x 85 Hz

 Current Resolution  : 800 x 600 pixels
 Current Depth       : 16-bit
 Current Refresh     : 60 Hz
 Current Rotation    : 0-degrees
 Current Pannable    : 800 x 600 pixels
 Current Rectangle   : (0,0)-(800,600)
 TV Connectors       : 0x00000003 COMPOSITE SVIDEO
 TV Connector In Use : 0x00000000
```

A

CODE SAMPLES

NvGetDisplayInfo.c

```
//------------------------------------------------------------------------
// NvGetDisplayInfo.c: NVCPL API example
//------------------------------------------------------------------------
// This example program demonstrates the use of the following APIs:
//  NvGetDisplayInfo
//  NvEnumDisplaySettings
//  EnumDisplayDevices
//  NvSelectDisplayDevice
//  NvRefreshConnectedDevices
//  NvGetConnectedDevicesString
//  NvGetActiveDevicesString
//------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <math.h>
#include "NvPanelApi.h"
```

```
//---------------------------------------------------------------------
// Defines
//---------------------------------------------------------------------

#define CMPERINCH   2.54f       // Number of centimeters per inch

#ifndef EDS_RAWMODE
#define EDS_RAWMODE 0x00000002  // Enumerate graphics card display modes not supported
by monitor
#endif//EDS_RAWMODE



//---------------------------------------------------------------------
// Typedefs
//---------------------------------------------------------------------

typedef BOOL (WINAPI* fEnumDisplayDevicesA)(LPCSTR lpDevice, DWORD iDevNum,
PDISPLAY_DEVICE lpDisplayDevice, DWORD dwFlags);



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

int PerformRefreshConnectedDevices(DWORD dwFlags);
int PerformNvGetDisplayInfo(int argc, char* argv[]);
int PerformNvEnumDisplaySettings(int argc, char* argv[]);
int PerformEnumConnectedDevices(int argc, char* argv[]);
BOOL IsEnumDisplayDevicesSupported();

//---------------------------------------------------------------------
// Functions
//---------------------------------------------------------------------

//---------------------------------------------------------------------
// Function:    main
// Description: Program entry-point.
// Parameters:  argc - Command-line argument count
//              argv - Command-line argument strings
// Returns:     0    - Success
//              else - Failure
//-----------------------------------------------------------------------
int main(int argc, char* argv[])
{
    if (argc < 2)
    {
        fprintf(stderr, "\n");
        fprintf(stderr, "To excercise the NvGetDisplayInfo API:\n"
                        "  Usage: %s [display#] [devicemoniker]\n"
                        "  Example: %s 1\n", argv[0], argv[0]);

        fprintf(stderr, "\n");
        fprintf(stderr, "To enumerate available display modes (constrained to monitor
capabilities) using the APIs:\n"
                        "  Usage: %s [modes|rawmodes] [standard|clone|hspan|vspan]
[display#] [devicemoniker] [devicemoniker]\n"
                        "  Example: %s modes clone 1 aa ab\n", argv[0], argv[0]);

        fprintf(stderr, "\n");
        fprintf(stderr, "To enumerate available devices using the APIs:\n"
                        "  Usage: %s enum\n"
                        "  Example: %s enum\n", argv[0], argv[0]);

        return 0;
    }

    // Refresh connected devices information
    if (PerformRefreshConnectedDevices(NVREFRESH_NONINTRUSIVE) != 0) // lightweight
refresh - some information may remain stale since last device scan
    {
        fprintf(stderr, "Failed to refresh connected devices.\n");
        return -1;
    }

    // Process command-line options
    if (stricmp(argv[1], "enum") == 0)
    {
        // Perform enumeration of connected devices
        if (PerformEnumConnectedDevices(argc, argv) != 0)
        {
            fprintf(stderr, “Failed to enumerate connected devices.\n”);
            return -1;
        }
    }
    else if ((stricmp(argv[1], “modes”   ) == 0) ||
             (stricmp(argv[1], “rawmodes”) == 0)  )
    {
        // Perform enumeration of display modes
        if (PerformNvEnumDisplaySettings(argc, argv) != 0)
        {
            fprintf(stderr, “Failed to perform NvEnumDisplaySettings.\n”);
            return -1;
        }
    }
    else
    {
        // Perform NvGetDisplayInfo
        if (PerformNvGetDisplayInfo(argc, argv) != 0)
        {
            fprintf(stderr, “Failed to perform NvGetDisplayInfo.\n”);
            return -1;
        }
    }

    return 0;
}

//-----------------------------------------------------------------------
// Function:    PerformRefreshConnectedDevices
// Description: Excercise the NvCplRefreshConnectedDevices() API.
// Parameters:  dwFlags - NVREFRESH_* flags
// Returns:     0       - Success
//              else    - Failure
//-----------------------------------------------------------------------
int PerformRefreshConnectedDevices(DWORD dwFlags)
{
    HMODULE                       hNvCplLib                      = NULL;
    fNvCplRefreshConnectedDevices NvCplRefreshConnectedDevices   = NULL;

    // Load NvCpl.dll
    hNvCplLib = LoadLibrary("NvCpl.dll");
    if (hNvCplLib == NULL)
    {
        fprintf(stderr, "Failed to locate NvCpl.dll.\n");
        return -1;
    }

    // Bind to NvCpl.dll functions
    NvCplRefreshConnectedDevices = (fNvCplRefreshConnectedDevices)
GetProcAddress(hNvCplLib, "NvCplRefreshConnectedDevices");
    if (NvCplRefreshConnectedDevices == NULL)
    {
        fprintf(stderr, "Failed to bind to NvCplRefreshConnectedDevices\n");
        return -1;
    }

    // Refresh connected devices information
    if (!NvCplRefreshConnectedDevices(dwFlags | NVREFRESH_SYSTEMWIDE))
    {
        fprintf(stderr, "Failed to refresh connected devices.\n");
        return -1;
    }

    return 0;
}

//----------------------------------------------------------------------
// Function:    PerformNvGetDisplayInfo
// Description: Excercise the NvGetDisplayInfo() API.
// Parameters:  argc - Command-line argument count
//              argv - Command-line argument strings
// Returns:     0    - Success
//              else - Failure
//----------------------------------------------------------------------
int PerformNvGetDisplayInfo(int argc, char* argv[])
{
    char               szDeviceMoniker[1024] = {0};
    HMODULE            hNvCplLib             = NULL;
    fNvGetDisplayInfo  NvGetDisplayInfo      = NULL;
    NVDISPLAYINFO      displayInfo           = {0};
    int                i                     = -1;

    // Concatenate argument list into single string
    // argv[1..n] = "[display#] [device moniker]"
    strcpy(szDeviceMoniker, argv[1]);
    for (i = 2; i < argc; i++)
    {
        strcat(szDeviceMoniker, " ");
        strcat(szDeviceMoniker, argv[i]);
    }

    // Load NvCpl.dll
    hNvCplLib = LoadLibrary("NvCpl.dll");
    if (hNvCplLib == NULL)
    {
        fprintf(stderr, "Failed to locate NvCpl.dll.\n");
        return -1;
    }

    // Bind to NvCpl.dll functions
    NvGetDisplayInfo = (fNvGetDisplayInfo) GetProcAddress(hNvCplLib,
"NvGetDisplayInfo");
    if (NvGetDisplayInfo == NULL)
    {
        fprintf(stderr, "Failed to bind to NvGetDisplayInfo.\n");
        return -1;
    }

    printf("Evaluating specified display...\n\n");

    // Get display information for specified device moniker
    //  displayInfo.cbSize must be set to size of structure
    //  displayInfo.dwInputFields1 must be set before call to indicate which fields to
retrieve
    //  displayInfo.dwOutputFields1 will be set on return to indicate which fields were
successfully retrived
```

```
    //   see NVDISPLAYINFO1_* bit definitions for field information, use 0xffffffff to
retrieve all fields
    memset(&displayInfo, 0, sizeof(displayInfo));
    displayInfo.cbSize = sizeof(displayInfo);
    displayInfo.dwInputFields1 = 0xffffffff; // 0xffffffff means all fields should be
retrieved
    displayInfo.dwInputFields2 = 0xffffffff; // 0xffffffff means all fields should be
retrieved
    if (!NvGetDisplayInfo(szDeviceMoniker, &displayInfo))
    {
        fprintf(stderr, “Failed to retrieve display info for device moniker \”%s\”.\n”,
szDeviceMoniker);
        return 2;
    }

    // Dump display device information to stdout
    printf(“ Display Number      : %ld\n”, displayInfo.dwWindowsMonitorNumber);
    if ((displayInfo.dwOutputFields2 & NVDISPLAYINFO2_CONNECTEDMONIKER) != 0) // not
supported by all drivers
    {
        printf(“ Connected Moniker   : %s\n”, (displayInfo.szConnectedMoniker[0] != ‘\
0’) ? displayInfo.szConnectedMoniker : “Not Connected”);
    }
    if ((displayInfo.dwOutputFields2 & NVDISPLAYINFO2_ACTIVEMONIKER) != 0) // not
supported by all drivers
    {
        printf(“ Active Moniker      : %s\n”, (displayInfo.szActiveMoniker[0] != ‘\0’)
? displayInfo.szActiveMoniker : “Not Active”);
    }
    printf(“\n”);

    printf(“ Display Adapter     : \”%s\”\n”, displayInfo.szAdapterName);
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_BOARDTYPE) != 0) // not supported
by all drivers
    {
        printf(“ Display Board       : “);
        switch (displayInfo.dwBoardType)
        {

            case NVBOARDTYPE_GEFORCE:
                printf(“GeForce”);
                break;

            case NVBOARDTYPE_QUADRO:
                printf("Quadro");
                break;

            case NVBOARDTYPE_NVS:
                printf("NVS");
                break;

            default:
                printf("0x%08lX", displayInfo.dwBoardType);
                break;
        }
        printf("\n");
    }
    printf(" Display Device      : \"%s\"\n", displayInfo.szWindowsDeviceName);
    printf(" Display Driver      : \"%s\"\n", displayInfo.szDriverVersion);
    printf(" Display Mode        : ");
    switch (displayInfo.nDisplayMode)
    {
        case NVDISPLAYMODE_STANDARD:
            printf("Single-Display");
            break;

        case NVDISPLAYMODE_CLONE:
            printf("Clone");
            break;

        case NVDISPLAYMODE_HSPAN:
            printf("Horizontal Span");
            break;

        case NVDISPLAYMODE_VSPAN:
            printf("Vertical Span");
            break;

        case NVDISPLAYMODE_DUALVIEW:
            printf("DualView");
            if (displayInfo.bDisplayIsPrimary)
            {
                printf(" (Primary)");
            }
            else
            {
                printf(" (Not Primary)");
            }
            break;

        default:
            printf("%d", displayInfo.nDisplayMode);
            break;
    }
    printf("\n");
    printf(" Display Qualifier   : %ld", displayInfo.dwWindowsMonitorNumber);
    switch (displayInfo.nDisplayMode)  // show multiple head qualifier?
    {
        case NVDISPLAYMODE_CLONE:
        case NVDISPLAYMODE_HSPAN:
        case NVDISPLAYMODE_VSPAN:
            if (displayInfo.bDisplayIsPrimary)
            {
                printf("a");
            }
            else
            {
                printf("b");
            }
            break;

        case NVDISPLAYMODE_STANDARD:
        case NVDISPLAYMODE_DUALVIEW:
        default:
            // do nothing
            break;
    }
    printf("\n");
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_DISPLAYHEADINDEX) != 0) // not
supported for inactive displays
    {
        printf(" Display Head        : %d\n", displayInfo.nDisplayHeadIndex);
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_DISPLAYNAME) != 0) // not
supported for inactive displays
    {
        printf(" Display Name        : \"%s\"\n", displayInfo.szDisplayName);
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_DISPLAYINSTANCECOUNT) != 0) //
not supported by all displays
    {
        printf(" Display Instance    : %d of %d\n", displayInfo.dwDisplayInstance,
displayInfo.dwDisplayInstanceCount);
    }
    if ((displayInfo.dwOutputFields2 & NVDISPLAYINFO2_PRODUCTNAME) != 0) // not
supported by all displays/drivers
    {
        printf(" Display Product     : \"%s\"\n", displayInfo.szProductName);
    }
    if ((displayInfo.dwOutputFields2 & NVDISPLAYINFO2_CUSTOMNAME) != 0) // not
supported by all displays/drivers
    {
        printf(" Display Custom      : \"%s\"\n", displayInfo.szCustomName);
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_VENDORNAME) != 0) // not
supported by all displays
    {
        printf(" Display Vendor      : \"%s\"\n", displayInfo.szVendorName);
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_MODELNAME) != 0) // not supported
by all displays
    {
        printf(" Display Model       : \"%s\"\n", displayInfo.szModelName);
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_UNIQUEID) != 0) // not supported
by all displays
    {
        printf(" Display Identifier  : 0x%08lX\n", displayInfo.dwUniqueId);
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_GENERICNAME) != 0) // not 
supported for inactive displays
    {
        printf(" Display Generic     : \"%s\"\n", displayInfo.szGenericName);
    }
    printf(" Display Type        : ");
    if (displayInfo.nDisplayType == NVDISPLAYTYPE_NONE)
    {
        printf("None");
    }
    else
    {
        switch (displayInfo.nDisplayType & NVDISPLAYTYPE_CLASS_MASK)
        {
            case NVDISPLAYTYPE_CRT:
                if (displayInfo.nDisplayType == NVDISPLAYTYPE_CRT)
                {
                    printf("Cathode Ray Tube (CRT)");
                }
                else
                {
                    printf("Cathode Ray Tube (CRT) [subtype: 0x%04X]", 
displayInfo.nDisplayType);
                }
                break;

            case NVDISPLAYTYPE_DFP:
                if (displayInfo.nDisplayType == NVDISPLAYTYPE_DFP)
                {
                    printf("Digital Flat Panel (DFP)");
                }
                else if (displayInfo.nDisplayType == NVDISPLAYTYPE_DFP_LAPTOP)
                {
                    printf("Laptop Display Panel");
                }
                else
                {
                    printf("Digital Flat Panel (DFP) [subtype: 0x%04X]", 
displayInfo.nDisplayType);
                }
                break;

            case NVDISPLAYTYPE_TV:
                if (displayInfo.nDisplayType == NVDISPLAYTYPE_TV)
                {
                    printf("Television");
                }
                else if (displayInfo.nDisplayType == NVDISPLAYTYPE_TV_HDTV)
                {
                    printf("High-Definition Television (HDTV)");
                }
                else
                {
                    printf("Television [subtype: 0x%04X]", displayInfo.nDisplayType);
                }
                break;

            default:
                printf("0x%04X", displayInfo.nDisplayType);
                break;
        }
    }
    printf("\n");

    if (((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_DISPLAYWIDTH ) != 0) && // not
supported by all displays
        ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_DISPLAYHEIGHT) != 0))
    {
        // note: display size here is maximum visible display surface area,
        //       reported monitor size is approximation of actual tube size
        printf(" Display Size        : %ld x %ldmm (approximately %.1f\")\n",
               displayInfo.mmDisplayWidth, displayInfo.mmDisplayHeight,
               (float) sqrt((((float) displayInfo.mmDisplayWidth ) *
displayInfo.mmDisplayWidth) +
                           (((float) displayInfo.mmDisplayHeight) *
displayInfo.mmDisplayHeight))/CMPERINCH/100);
    }

    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_GAMMACHARACTERISTIC) != 0) // not
supported by all displays
```

```
    {
        printf(" Display Transfer    : %.02f\n", displayInfo.fGammaCharacteristic);
    }

    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_OPTIMALMODE) != 0) // not
supported by inactive displays
    {
        printf(" Display Optimal     : %ld x %ld x %ld Hz\n",
displayInfo.dwOptimalPelsWidth,

displayInfo.dwOptimalPelsHeight,

displayInfo.dwOptimalDisplayFrequency);
    }

    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_MAXIMUMSAFEMODE) != 0) // not
supported by inactive displays
    {
        printf(" Display Largest Safe: %ld x %ld x %ld Hz\n",
displayInfo.dwMaximumSafePelsWidth,

displayInfo.dwMaximumSafePelsHeight,

displayInfo.dwMaximumSafeDisplayFrequency);
    }

    printf("\n");

    // Dump current mode information to stdout
    printf(" Current Resolution  : %ld x %ld pixels\n"   ,
displayInfo.dwVisiblePelsWidth, displayInfo.dwVisiblePelsHeight);
    printf(" Current Depth       : %ld-bit\n"            , displayInfo.dwBitsPerPel);
    printf(" Current Refresh     : %ld Hz\n"             ,
displayInfo.dwDisplayFrequency);
    printf(" Current Rotation    : %ld-degrees\n"        ,
displayInfo.dwDegreesRotation);
    printf(" Current Pannable    : %ld x %ld pixels\n"   , displayInfo.dwPelsWidth,
displayInfo.dwPelsHeight);
    printf(" Current Rectangle   : (%ld,%ld)-(%ld,%ld)\n",
displayInfo.rcDisplayRect.left,

displayInfo.rcDisplayRect.top,
displayInfo.rcDisplayRect.right,

displayInfo.rcDisplayRect.bottom);
   if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_TVCONNECTORTYPES) != 0) // not
supported by all displays
   {
        printf(" TV Connectors       : 0x%08lX", displayInfo.dwTVConnectorTypes);
        if ((displayInfo.dwTVConnectorTypes & NVTVCONNECTOR_UNKNOWN) != 0)
        {
            printf(" UNKNOWN");
        }
        if ((displayInfo.dwTVConnectorTypes & NVTVCONNECTOR_COMPOSITE) != 0)
        {
            printf(" COMPOSITE");
        }
        if ((displayInfo.dwTVConnectorTypes & NVTVCONNECTOR_SVIDEO) != 0)
        {
            printf(" SVIDEO");
        }
        if ((displayInfo.dwTVConnectorTypes & NVTVCONNECTOR_COMPONENT) != 0)
        {
            printf(" COMPONENT");
        }
        if ((displayInfo.dwTVConnectorTypes & NVTVCONNECTOR_EIAJ4120) != 0)
        {
            printf(" EIAJ4120");
        }
        if ((displayInfo.dwTVConnectorTypes & NVTVCONNECTOR_EIAJ4120CVBSBLUE) != 0)
        {
            printf(" EIAJ4120CVBS");
        }
        if ((displayInfo.dwTVConnectorTypes & NVTVCONNECTOR_SCART) != 0)
        {
            printf(" SCART");
        }
        printf("\n");
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_CURRENTCONNECTORTYPE) != 0) //
not supported by all displays
```

```
    {
        printf(" TV Connector In Use : 0x%08lX\n", displayInfo.dwCurrentConnectorType);
        if ((displayInfo.dwCurrentConnectorType & NVTVCONNECTOR_UNKNOWN) != 0)
        {
            printf(" UNKNOWN");
        }
        if ((displayInfo.dwCurrentConnectorType & NVTVCONNECTOR_COMPOSITE) != 0)
        {
            printf(" COMPOSITE");
        }
        if ((displayInfo.dwCurrentConnectorType & NVTVCONNECTOR_SVIDEO) != 0)
        {
            printf(" SVIDEO");
        }
        if ((displayInfo.dwCurrentConnectorType & NVTVCONNECTOR_COMPONENT) != 0)
        {
            printf(" COMPONENT");
        }
        if ((displayInfo.dwCurrentConnectorType & NVTVCONNECTOR_EIAJ4120) != 0)
        {
            printf(" EIAJ4120");
        }
        if ((displayInfo.dwCurrentConnectorType & NVTVCONNECTOR_EIAJ4120CVBSBLUE) != 0)
        {
            printf(" EIAJ4120CVBS");
        }
        if ((displayInfo.dwCurrentConnectorType & NVTVCONNECTOR_SCART) != 0)
        {
            printf(" SCART");
        }
        printf("\n");
    }
    if ((displayInfo.dwOutputFields2 & NVDISPLAYINFO2_DVIOVERHDTV) != 0) // not 
supported by all drivers
    {
        printf(" Use DVI as HDTV     : %s\n", (displayInfo.bDVIOverHDTV) ? "Active" : 
"Not active");
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_TVFORMAT) != 0) // not supported 
by all displays
```

```
    {
        printf(" Current TV Format   : ");
        switch (displayInfo.nTvFormat)
        {
            case NVTVFORMAT_NONE:
                printf("Not applicable");
                break;

            case NVTVFORMAT_NTSC_M:
                printf("NTSC/M");
                break;

            case NVTVFORMAT_NTSC_J:
                printf("NTSC/J");
                break;

            case NVTVFORMAT_PAL_M:
                printf("PAL/M");
                break;

            case NVTVFORMAT_PAL_A:
                printf("PAL/BDGHI");
                break;

            case NVTVFORMAT_PAL_N:
                printf("PAL/N");
                break;

            case NVTVFORMAT_PAL_NC:
                printf("PAL/NC");
                break;

            case NVTVFORMAT_HD480i:
                printf("HDTV 480i");
                break;

            case NVTVFORMAT_HD480p:
                printf("HDTV 480p");
                break;
            case NVTVFORMAT_HD720p:
                printf("HDTV 720p");
                break;

            case NVTVFORMAT_HD1080i:
                printf("HDTV 1080i");
                break;

            case NVTVFORMAT_HD1080p:
                printf("HDTV 1080p");
                break;

            default:
                printf("%d", displayInfo.nTvFormat);
                break;
        }
        printf("\n");
    }
    if ((displayInfo.dwOutputFields1 & NVDISPLAYINFO1_DFPSCALING) != 0) // not
supported by all displays
    {
        printf(" Current DFP Scaling : ");
        switch (displayInfo.nDfpScaling)
        {
            case NVDFPSCALING_NONE:
                printf("Not applicable");
                break;

            case NVDFPSCALING_NATIVE:
                printf("Monitor Native");
                break;

            case NVDFPSCALING_SCALED:
                printf("Scaling");
                break;

            case NVDFPSCALING_CENTERED:
                printf("Centering");
                break;

            case NVDFPSCALING_SCALEDASPECT:
                printf("Scaling (Fixed Aspect Ratio)");
                break;

            default:
                printf("%d", displayInfo.nDfpScaling);
                break;
        }
        printf("\n");
    }

    return 0;
}

//-----------------------------------------------------------------------
// Function:    PerformNvEnumDisplaySettings
// Description: Excercise the NvEnumDisplaySettings() API.
// Parameters:  argc - Command-line argument count
//              argv - Command-line argument strings
// Returns:     0    - Success
//              else - Failure
//-----------------------------------------------------------------------
int PerformNvEnumDisplaySettings(int argc, char* argv[])
{
    char                     szDeviceMoniker[1024] = {0};
    HMODULE                  hNvCplLib             = NULL;
    fNvEnumDisplaySettings   NvEnumDisplaySettings = NULL;
    fNvGetLastErrorMessageA  NvGetLastErrorMessage = NULL;
    NVRESULT                 nr                    = NV_OK;
    NVDISPLAYMODE            displayMode           = NVDISPLAYMODE_NONE;
    DEVMODE*                 pDevModes             = NULL;
    DWORD                    dwNumDevModes         =  0;
    DWORD                    dwFlags               =  0;
    DWORD                    dwDevModeIndex        =  0;
    int                      i                     = -1;

    // Concatenate argument list into single string
    // argv[1] = “modes” or “rawmodes”
    // argv[2] = “standard|clone|hspan|vspan”
    // argv[3..n] = “[display#] [device moniker] [device moniker]”

    if (argc < 4)
    {
        return -1;
    }

    if (stricmp(argv[1], “rawmodes”) == 0)
    {
        dwFlags = EDS_RAWMODE; // include adapter modes regardless of ability of
monitors to display them
    }

    if (stricmp(argv[2], “standard”) == 0)
    {
        displayMode = NVDISPLAYMODE_STANDARD;
    }
    else if (stricmp(argv[2], “clone”) == 0)
    {
        displayMode = NVDISPLAYMODE_CLONE;
    }
    else if (stricmp(argv[2], “hspan”) == 0)
    {
        displayMode = NVDISPLAYMODE_HSPAN;
    }
    else if (stricmp(argv[2], “vspan”) == 0)
    {
        displayMode = NVDISPLAYMODE_VSPAN;
    }
    else
    {
        fprintf(stderr, “Unexpected NVDISPLAYMODE \”%s\”\n”, argv[2]);
        return -1;
    }

    strcpy(szDeviceMoniker, argv[3]);
    for (i = 4; i < argc; i++)
    {
        strcat(szDeviceMoniker, " ");
        strcat(szDeviceMoniker, argv[i]);
    }

    // Load NvCpl.dll
    hNvCplLib = LoadLibrary("NvCpl.dll");
    if (hNvCplLib == NULL)
    {
        fprintf(stderr, "Failed to locate NvCpl.dll.\n");
        return -1;
    }

    // Bind to NvCpl.dll functions
    NvEnumDisplaySettings = (fNvEnumDisplaySettings) GetProcAddress(hNvCplLib,
"NvEnumDisplaySettings");
    if (NvEnumDisplaySettings == NULL)
    {
        fprintf(stderr, "Failed to bind to NvEnumDisplaySettings.\n");
        return -1;
    }

    NvGetLastErrorMessage = (fNvGetLastErrorMessageA) GetProcAddress(hNvCplLib,
"NvGetLastErrorMessageA");
    if (NvGetLastErrorMessage == NULL)
    {
        fprintf(stderr, "Failed to bind to NvGetLastErrorMessageA.\n");
        return -1;
    }

    printf("Enumerating available modes for specified display%s...\n\n", (dwFlags &
EDS_RAWMODE) ? " (raw modes)" : "");

    // Get number of display modes for display combinatioon
    dwNumDevModes = 0L;
    nr = NvEnumDisplaySettings(szDeviceMoniker,
                               displayMode,
                               0L,
                               NULL,
                               &dwNumDevModes,
                              dwFlags);
    if ((nr != NV_OK) &&
        (nr != NV_OUTOFMEMORY))  // dwNumDevModes was zero on input, so should always
have at least one mode
    {
        fprintf(stderr, "Failed to count display modes: %s\n",
NvGetLastErrorMessage());
        return 2;
    }

    // Allocate storage for display modes
    pDevModes = (DEVMODE*) malloc(dwNumDevModes * sizeof(DEVMODE));
    if (pDevModes == NULL)
    {
        fprintf(stderr, "Out of memory for display modes.\n");
        return -1;
    }

    // Get display modes for display combination
    nr = NvEnumDisplaySettings(szDeviceMoniker,
                              displayMode,
                              sizeof(DEVMODE),
                              pDevModes,
                              &dwNumDevModes,
                              dwFlags);
    if (nr != NV_OK)
    {
        fprintf(stderr, "Failed to enumerate display modes: %s\n",
NvGetLastErrorMessage());
        return 2;
    }

    // Dump display modes information to stdout
    for (dwDevModeIndex = 0;
         dwDevModeIndex < dwNumDevModes; dwDevModeIndex++)
    {
        printf(" Display Mode %03lu: %ld x %ld x %ld x %ld Hz",
            dwDevModeIndex,
            pDevModes[dwDevModeIndex].dmPelsWidth,
            pDevModes[dwDevModeIndex].dmPelsHeight,
            pDevModes[dwDevModeIndex].dmBitsPerPel,
            pDevModes[dwDevModeIndex].dmDisplayFrequency);
        if (pDevModes[dwDevModeIndex].dmDisplayFlags != 0)
        {
            printf(" - Flags=0x%08lX", pDevModes[dwDevModeIndex].dmDisplayFlags);
        }
        printf("\n");
    }

    // Free storage for display modes
    free(pDevModes);
    pDevModes = NULL;

    return 0;
}

//-----------------------------------------------------------------------
// Function:    PerformEnumConnectedDevices
// Description: Excercise the EnumDisplayDevices() and
//              NvCplGetConnected/ActiveDevicesString() APIs.
// Parameters:  argc - Command-line argument count
//              argv - Command-line argument strings
// Returns:     0    - Success
//              else - Failure
//-----------------------------------------------------------------------
int PerformEnumConnectedDevices(int argc, char* argv[])
{
    HMODULE                            hUser32Lib                         = NULL;
    fEnumDisplayDevicesA               EnumDisplayDevicesA                = NULL;
    DISPLAY_DEVICEA                    displayDeviceA                     = {0};
    int                                nEnumDisplayIndex                  = -1;
    int                                nWindowsMonitorNumber              = -1;

    HMODULE                            hNvCplLib                          = NULL;
    fNvSelectDisplayDevice             NvSelectDisplayDevice              = NULL;
    fNvCplGetRealConnectedDevicesString NvCplGetRealConnectedDevicesString = NULL;
    fNvCplGetActiveDevicesString       NvCplGetActiveDevicesString        = NULL;
    fNvCplGetMSOrdinalDeviceString     NvCplGetMSOrdinalDeviceString      = NULL;
    char                                szConnectedDevices[1024]           = {0};

    // Load NvCpl.dll
    hNvCplLib = LoadLibrary("NvCpl.dll");
    if (hNvCplLib == NULL)
    {
        fprintf(stderr, "Failed to locate NvCpl.dll.\n");
        return -1;
    }

    // Bind to NvCpl.dll functions
    NvSelectDisplayDevice = (fNvSelectDisplayDevice) GetProcAddress(hNvCplLib, 
"NvSelectDisplayDevice");
    if (NvSelectDisplayDevice == NULL)
    {
        fprintf(stderr, "Failed to bind to NvSelectDisplayDevice\n");
        return -1;
    }

    NvCplGetRealConnectedDevicesString = (fNvCplGetRealConnectedDevicesString) 
GetProcAddress(hNvCplLib, "NvCplGetRealConnectedDevicesString");
    if (NvCplGetRealConnectedDevicesString == NULL)
    {
        fprintf(stderr, "Failed to bind to NvCplGetRealConnectedDevicesString\n");
        return -1;
    }

    NvCplGetActiveDevicesString = (fNvCplGetActiveDevicesString) 
GetProcAddress(hNvCplLib, "NvCplGetActiveDevicesString");
    if (NvCplGetActiveDevicesString == NULL)
    {
        fprintf(stderr, "Failed to bind to NvCplGetActiveDevicesString\n");
        return -1;
    }

    NvCplGetMSOrdinalDeviceString = (fNvCplGetMSOrdinalDeviceString) 
GetProcAddress(hNvCplLib, "NvCplGetMSOrdinalDeviceString");
    if (NvCplGetMSOrdinalDeviceString == NULL)
    {
        fprintf(stderr, "Failed to bind to NvCplGetMSOrdinalDeviceString\n");
        return -1;
    }

    printf("Enumerating available displays...\n\n");

    // Check if Windows native multiple monitor support is available
    if (IsEnumDisplayDevicesSupported())
    {
        // Load USER32.dll
        hUser32Lib = GetModuleHandle("USER32.dll");
        if (hUser32Lib == NULL)
        {
            fprintf(stderr, "Failed to locate USER32.dll\n");
            return -1;
        }

        // Bind to USER32.dll functions
        EnumDisplayDevicesA = (fEnumDisplayDevicesA) GetProcAddress(hUser32Lib,
"EnumDisplayDevicesA");
        if (EnumDisplayDevicesA == NULL)
        {
            fprintf(stderr, "Failed to bind to EnumDisplayDevicesA\n");
            return -1;
        }

        // Enumerate displays
        nWindowsMonitorNumber  = 1;
        for (nEnumDisplayIndex = 0;
             nEnumDisplayIndex < 256; nEnumDisplayIndex++)
        {
            // Get next display device from Windows
            ZeroMemory(&displayDeviceA, sizeof(displayDeviceA));
            displayDeviceA.cb = sizeof(displayDeviceA);
            if (!EnumDisplayDevicesA(NULL, nEnumDisplayIndex, &displayDeviceA, 0L))
            {
                break; // no more monitors attached, so abort search
            }

            // Process actual display devices
            if ((displayDeviceA.StateFlags & DISPLAY_DEVICE_MIRRORING_DRIVER) !=
DISPLAY_DEVICE_MIRRORING_DRIVER)
            {
                printf(" Display %d [%s - %s]\n", nWindowsMonitorNumber,
displayDeviceA.DeviceName, displayDeviceA.DeviceString);

                // Check if display attached to desktop (Windows OS only allows heads
attached to the desktop be communicated with)
               if ((displayDeviceA.StateFlags & DISPLAY_DEVICE_ATTACHED_TO_DESKTOP) ==
DISPLAY_DEVICE_ATTACHED_TO_DESKTOP)
                {
                    // Select display device
                    if (!NvSelectDisplayDevice(nWindowsMonitorNumber))
                    {
                        fprintf(stderr, "Failed to select device for Windows monitor
number %d\n", nWindowsMonitorNumber);
                        return -1;
                    }
                    printf("\n");

                    // Get display heads information
                    printf("  Relative to connected devices,\n"
                           "    where \"#AA\" is first connected device.\n");
                    if (NvCplGetRealConnectedDevicesString(szConnectedDevices,
sizeof(szConnectedDevices), FALSE/*bOnlyActive*/) == TRUE)
                    {
                        printf("    * Connected Devices          : \"%s\"\n",
szConnectedDevices);
                    }
                    if (NvCplGetRealConnectedDevicesString(szConnectedDevices,
sizeof(szConnectedDevices), TRUE/*bOnlyActive*/) == TRUE)
                    {
                        printf("    * Connected Devices (Active) : \"%s\"\n",
szConnectedDevices);
                    }
                    printf("\n");
                    printf("  Relative to active devices,\n"
                           "    where \"AA\" is first active device\n"
                           "    and \"-AA\" means no connection.\n");
                    if (NvCplGetActiveDevicesString(szConnectedDevices,
sizeof(szConnectedDevices)) == TRUE)
                    {
                        printf("    * Active Devices             : \"%s\"\n",
szConnectedDevices);
                    }
                    printf("\n");
                    printf(" Output from NvCplGetMSOrdinalDeviceString,\n"
                           "    where the first listed device is primary\n"
                           "    and the later is secondary if shown.\n") ;
                    if (NvCplGetMSOrdinalDeviceString(nWindowsMonitorNumber,
szConnectedDevices, sizeof(szConnectedDevices)) == TRUE)
                    {
                        printf("    * Primary+Secondary Devices  : \"%s\"\n",
szConnectedDevices);
                    }
                }
                else
                {
                    printf("  Not attached to desktop.\n");
                }
                printf("\n");

                // Increment Windows monitor number
                nWindowsMonitorNumber++;
            }
        }
    }
    else
    {
        // Get display heads information
        printf("  Relative to connected devices,\n"
               "    where \"#AA\" is first connected device.\n");
        if (NvCplGetRealConnectedDevicesString(szConnectedDevices,
sizeof(szConnectedDevices), FALSE/*bOnlyActive*/) == TRUE)
        {
            printf("    * Connected Devices          : \"%s\"\n", szConnectedDevices);
        }
        if (NvCplGetRealConnectedDevicesString(szConnectedDevices,
sizeof(szConnectedDevices), TRUE/*bOnlyActive*/) == TRUE)
        {
            printf("    * Connected Devices (Active) : \"%s\"\n", szConnectedDevices);
        }
        printf("\n");
        printf("  Relative to active devices,\n"
               "    where \"AA\" is first active device\n"
               "    and \"-AA\" means no connection.\n");
        if (NvCplGetActiveDevicesString(szConnectedDevices, sizeof(szConnectedDevices))
== TRUE)
        {
            printf("    * Active Devices             : \"%s\"\n", szConnectedDevices);
        }
        printf("\n");
        printf(" Output from NvCplGetMSOrdinalDeviceString,\n"
               "    where the first listed device is primary\n"
               "    and the later is secondary if shown.\n") ;
        if (NvCplGetMSOrdinalDeviceString(nWindowsMonitorNumber, szConnectedDevices,
sizeof(szConnectedDevices)) == TRUE)
        {
            printf("    * Primary+Secondary Devices  : \"%s\"\n", szConnectedDevices);
        }
        printf("\n");
    }
    printf("  Use NvGetDisplayInfo() to translate monikers that are relative \n"
           "  to connected devices to be relative to active devices and vice versa.\
n");

    return 0;
}

//----------------------------------------------------------------------
// Function:    IsEnumDisplayDevicesSupported
// Description: Determine if OS supports EnumDisplayDevices() Win32 API.
// Parameters:  .
// Returns:     TRUE - EnumDisplayDevices is supported
//              TRUE - EnumDisplayDevices is NOT supported
//----------------------------------------------------------------------
BOOL IsEnumDisplayDevicesSupported()
{
    BOOL bEnumDisplayDevicesSupported = FALSE;
    OSVERSIONINFO osvi = {0};
    osvi.dwOSVersionInfoSize = sizeof(osvi);
    if (GetVersionEx(&osvi))
    {
        switch (osvi.dwPlatformId)
        {
            case VER_PLATFORM_WIN32_WINDOWS:  // 9X series
                if  (((((osvi.dwBuildNumber >> 24) & 0x000000FF) ==  4) &&  // Win98 or
higher
                      (((osvi.dwBuildNumber >> 16) & 0x000000FF) >= 10)  )
                     || (((osvi.dwBuildNumber >> 24) & 0x000000FF) > 4))
                {
                    bEnumDisplayDevicesSupported = TRUE;
                }
                break;

            case VER_PLATFORM_WIN32_NT:       // NT series
                if (osvi.dwMajorVersion >= 5) //   Win2K or higher
                {
                    bEnumDisplayDevicesSupported = TRUE;
                }
                break;

            default:
                break;
        }
    }

    if (bEnumDisplayDevicesSupported)
    {
        HMODULE              hUser32Lib          = NULL;
        fEnumDisplayDevicesA EnumDisplayDevicesA = NULL;

        // Load USER32.dll
        hUser32Lib = GetModuleHandle("USER32.dll");
        if (hUser32Lib == NULL)
        {
            bEnumDisplayDevicesSupported = FALSE;
        }
        // Bind to USER32.dll functions
        else
        {
            EnumDisplayDevicesA = (fEnumDisplayDevicesA) GetProcAddress(hUser32Lib,
"EnumDisplayDevicesA");
            if (EnumDisplayDevicesA == NULL)
            {
                bEnumDisplayDevicesSupported = FALSE;
            }
        }
    }

    return bEnumDisplayDevicesSupported;
}
```