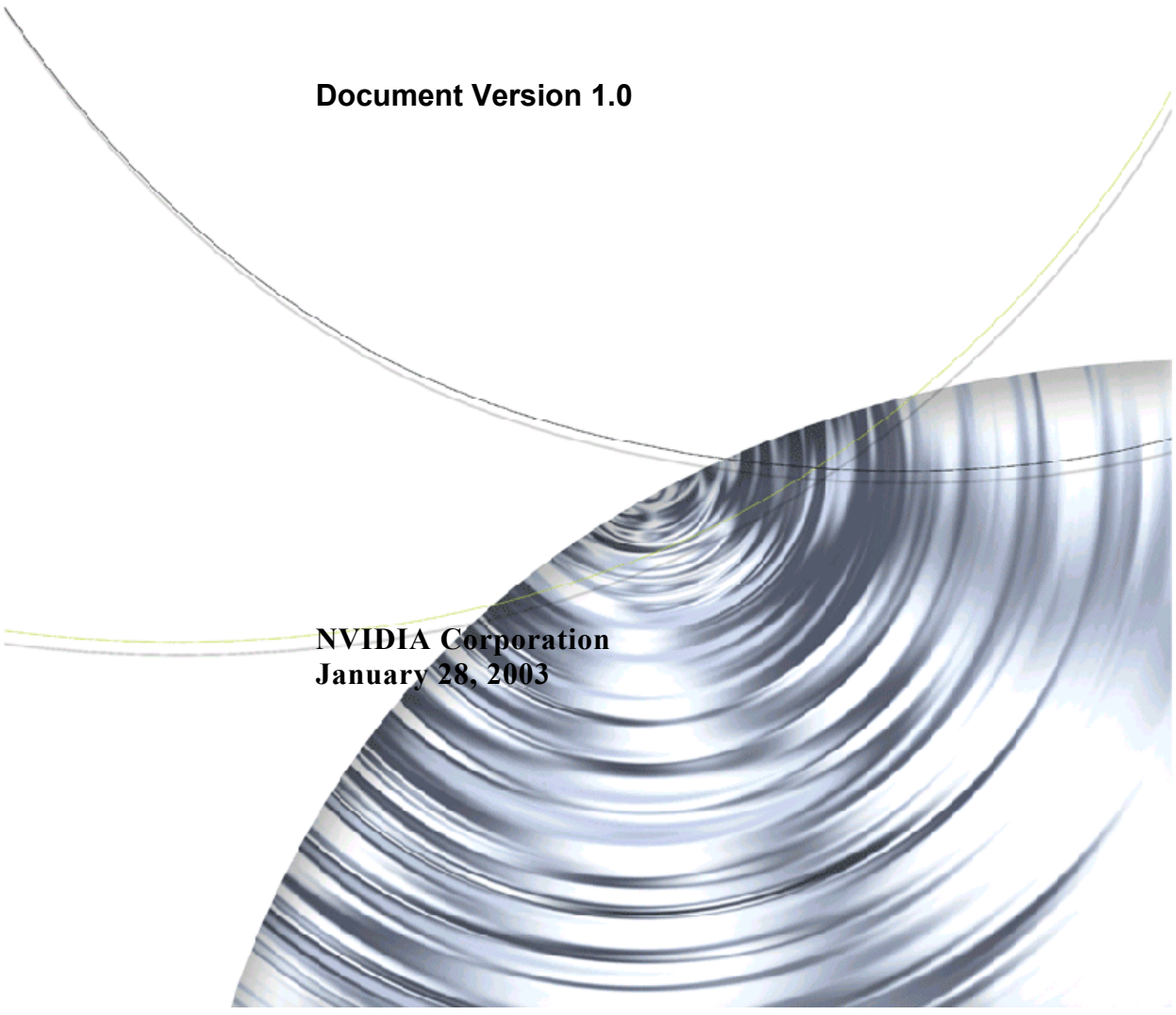
UTILITIES AND APIS

NVCPL.DLL API Manual

Document Version 1.0



NVIDIA Corporation
January 28, 2003

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 is divided into two sections:

- Desktop Configuration

  This chapter describes the command line function—`dtcfg` (Desktop Configuration)—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.

Document Revision History

Revision	Date	Description
1.0	1/28/03	Initial Release. Combined previous PowerMizer API with DTCFG document. Added new APIs.

System Requirements

The `NVCPL.DLL` APIs support the NVIDIA Detonator XP Release 40 drivers for Windows® 98, Windows Me, Windows 2000 and Windows XP.

2

DESKTOP CONFIGURATION

Overview of DTCFG

The NVIDIA Control Panel library exports a command line function—"**dtcfg**" (Desktop Configuration)—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.

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:

- "Setting Delay Times" on page 4 explains how to coordinate multiple commands.
- "Configuring the Desktop" on page 5 explains the command line format and describes each of the DTCFG commands.

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.

Configuring the Desktop

DTCFG Command Format

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 following:
  - 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
  - rotate - Rotate a display
  - setmode - Set the display mode on a display
  - setview - Set one of the TwinView modes on a display
  - 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.
  - setbrightness - Set the brightness for the desired color channel.
  - setscaling - Sets the scaling of a display.
- *display#* is the display number on the windows settings page.

  This can be any of the values shown on that 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.

Understanding the Command Options

Following are the formats for each <*command*> option in the desktop configuration command line.

attach

Description Attach a display to the desktop.

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 Detect devices 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.

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.

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).

setview

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

Format `rundll32.exe NvCpl.dll,dtcfg setview <display#> <viewtype> [<primary device mnemonic>] [<secondary device mnemonic>]`

where

- *viewtype* can be any of the following:
  - `standard` (or `normal`)
  - `clone`
  - `hspan`
  - `vspan`
- *primary device mnemonic* and *secondary device mnemonic* 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 will make the assignments automatically.

  To specify, use the following two character string format:
  - `A0–A7` (or `AA–AH`) for analog devices.

    These normally include CRTs or analog flat panels.
  - `D0–D7` (or `DA–DH`) for digital devices.

    These normally include digital flat panels and digital CRTs.
  - `T0–T7` (or `TA–TH`) for TVs.

  The first character is a mnemonic that specifies the display type—**`A`** corresponds to analog display, **`D`** corresponds to digital displays, and **`T`** corresponds to TVs.

  The second character specifies the display in one of two ways:

  A–H: For most purposes, use A-H, where A maps to the first display found of a given device type, regardless of where it is connected, B maps to the second display, and so on.

  0–7: Each physical connector is given a bit assignment, starting with bit 0 and progressing one bit higher for each additional connector within the same display type. Use 0-7 only if you know the actual connector assignments on the adapter card, otherwise the command will fail if there is no device on that connector.

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 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.

setdvc

Description Set the Digital Vibrance level of the specified display.

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

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

Note: This is currently the only command that supports "all" for the display number.

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.

setgamma

Description Set the gamma level of the specified display.

Format rundll32.exe NvCpl.dll,dtcfg setgamma <*display#*> <*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 through 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.

setcontrast

Description Set the contrast level of the specified display.

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

setbrightness

Description Set the brightness level of the specified display.

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

setscaling

Description Set the scaling of the specified display.

Format `rundll32.exe NvCpl.dll,dtcfg setscaling <display#> <mode: 0,1, 2, 3, 4, 5>`

Where the scaling modes are defined as follows:

0 : Default

1: Native

2: Scaled

3: Centered

4: 8 bit scaled

5: Aspect scaling (for wide panel LCD)

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

Set the scaling of display #1 to scaled mode.

3

CONTROL PANEL APIS

This chapter documents the following functions that are exported from `nvcpl.dll`:

API Descriptions

Gamma Ramp

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

- `NvColorGetGammaRamp()` gets the current gamma color values.
- `NvColorSetGammaRamp()` sets the gamma color values.

NvColorGetGammaRamp()

Function Call	`BOOL NvColorGetGammaRamp(LPTSTR szUserDisplay, PGAMMARAMP_MULTI pGammaNew);`
Parameters In	`LPTSTR szUserDisplay --` Either specify "all" for all displays, or specify a particular display using the following two character format: The first character specifies the display type—**`A`** corresponds to analog displays, **`D`** corresponds to digital displays, and **`T`** corresponds to TVs. The second character is one of A-H, where A maps to the first display found of a given device type, regardless of where it is connected, B maps to the second display, and so on.
Parameters Out	`PGAMMARAMP_MULTI pGammaNew --` the new gamma table values
Return Values	**`TRUE`** if the new gamma values have been applied. **`FALSE`** if the display name is not valid.

NvColorSetGammaRamp()

Function Call	`BOOL NvColorSetGammaRamp(LPTSTR szUserDisplay, DWORD dwUserRotateFlag, PGAMMARAMP_MULTI pGammaNew);`
Parameters In	`LPTSTR szUserDisplay --` Either specify "all" for all displays, or specify a particular display using the following two character format: The first character specifies the display type—**`A`** corresponds to analog displays, **`D`** corresponds to digital displays, and **`T`** corresponds to TVs. The second character is one of A-H, where A maps to the first display found of a given device type, regardless of where it is connected, B maps to the second display, and so on. `DWORD dwUserRotateFlag --` display rotation flag `PGAMMARAMP_MULTI pGammaNew --` 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.

Relevant Gamma Structures

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

// Multiple Head Display Gamma Support
typedef struct GAMMARAMP_MULTI {
    DWORD      dwMask;
    GAMMARAMP  grGammaRamp;
} GAMMARAMP_MULTI, *PGAMMARAMP_MULTI;
```

Code Sample

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

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

typedef struct GAMMARAMP_MULTI
{
  DWORD       dwMask;
  GAMMARAMP   grGammaRamp;
} GAMMARAMP_MULTI, *PGAMMARAMP_MULTI;

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


void main()
{
```

```
  HINSTANCE           hCpl = NULL;
  PCOLORGETGAMMARAMP  pGetGamma = NULL;
  PCOLORSETGAMMARAMP  pSetGamma = NULL;
  GAMMARAMP_MULTI     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( "all", &Gamma );

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

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

```
        return;
    }

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

    FreeLibrary( hCpl );
}
```

Get Windows Display Mode

The function `NvGetWindowsDisplayState()` returns the multimonitor state for the specified Windows display. The state is either Windows DualView, or one of the nView modes—Standard, Horizontal Spanning, Vertical Spanning, or Clone mode.

NvGetWindowsDisplayState()

Function Call	`int APIENTRY NvGetWindowsDisplayState(int iDisplayIndex);`
Parameters In	`iDisplayIndex` -- the display number shown on the Windows Display Properties->Settings page. A value of 0 indicates the current Windows primary display.
Return Values	`-1` -- The display is in an unknown view mode. `0` -- The call failed. `1` -- The requested display index does not exist. `2` -- The display is not attached to the desktop. `3` -- The display is attached to the desktop, but the driver is not an NVIDIA driver. `4` -- The display is in nView Standard mode (not in Dualview). `5` -- The display is in Dualview mode (not in nView Standard) `6` -- The display is in nView Clone mode. `7` -- The display is in nView Horizontal Spanning mode. `8` -- The display is in nView Vertical Spanning mode.

Get Connected Devices

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.

NvCplGetConnectDevicesString()

Function Call	`BOOL APIENTRY NvCplGetConnectedDevicesString(LPTSTR lpszTextBuffer, DWORD cbTextBuffer, BOOL bOnlyActive);`
Parameters In	`lpszTextBuffer` -- The buffer to receive the requested strings. `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.
Return Values	• If `bOnlyActive` is FALSE, then the list of all connected devices is returned. • If `bOnlyActive` is TRUE, then the list of active connected devices is returned.

Sample Code

The following is an example of how the Get Connected Devices API is used:

```
#include <stdio.h>
#include <windows.h>

typedef BOOL (APIENTRY *FNCGETCONNECTEDDEVICESSTRING)(LPSTR
lpszTextBuffer, DWORD cbTextBuffer, BOOL bOnlyActive);

int main()
{
  TCHAR connectedString[256];
  FNCGETCONNECTEDDEVICESSTRING funcNvCplGetConnectedDevicesString;
  HMODULE hCplDLL;

  hCplDLL = LoadLibrary("NvCpl.dll");

  if (hCplDLL == NULL)
  {
      printf("Unable to load library NvCpl.dll\n");
```

```
        return -1;
    }

    funcNvCplGetConnectedDevicesString =
(FNCGETCONNECTEDDEVICESSTRING)GetProcAddress(hCplDLL,
"NvCplGetConnectedDevicesString");
    if (funcNvCplGetConnectedDevicesString == NULL)
    {
       printf("Unable to find function NvCplGetConnectedDevicesString\n");
       FreeLibrary(hCplDLL);
       return -1;
    }

    // ok now for testing do a 2 character string which should fail.
    if (!(*funcNvCplGetConnectedDevicesString)(connectedString, 2, FALSE))
    {
       printf("Function failed like it should have\n");
    }

    // ok now for testing do a 5 character string which should fail,
    // because no room for \0
    if (!(*funcNvCplGetConnectedDevicesString)(connectedString, 5, FALSE))
    {
       printf("Function failed like it should have\n");
    }

    // ok now for testing do a 6 character string which should work
    if (!(*funcNvCplGetConnectedDevicesString)(connectedString, 6, FALSE))
    {
       printf("Function failed, but shouldn't have!\n");
       FreeLibrary(hCplDLL);
       return -2;
    }

    printf("Function succeeded, connected = %s\n", connectedString);

    // ok now for testing do a 6 character string which should work,
    // with active masks
    if (!(*funcNvCplGetConnectedDevicesString)(connectedString, 6, TRUE))
```

```
    {
        printf("Function failed, but shouldn't have!\n");
        FreeLibrary(hCplDLL);
        return -3;
    }

    printf("Function succeeded, active devices = %s\n", connectedString);

    return 0;
}
```

PowerMizer

The PowerMizer API provides functions that read and write the PowerMizer level to be used when a laptop is running on battery power. The API consists of two functions that are exported from `nvcpl.dll`:

- `nvGetPwrMzrLevel()` gets the current PowerMizer level.
- `nvSetPwrMzrLevel()` sets the current PowerMizer level.

There are three PowerMizer levels, and are described as follows:

PowerMizer Level[i]	Comments
1 (Maximum performance)	Automatically set when the laptop is running on AC power.
2 (Medium setting)	This setting requires that the laptop is running on battery power.
3 (Maximum power savings)	This setting requires that the laptop is running on battery power, and provides the maximum battery life.

i. A call to nvSetPwrMzrLevel() that sets a different level takes effect only after the laptop is switched to battery power.

nvGetPwrMzrLevel()

Function Call	`BOOL nvGetPwrMzrLevel(DWORD* dwLevel);`
Parameters In	`DWORD*` must be a valid pointer.
Return Values	**`True`** if the PowerMizer level is obtained successfully. **`False`** for the following reasons: • The `DWORD*` pointer passed in 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()

Function Call	`BOOL nvSetPwrMzrLevel(DWORD* dwLevel);`
Parameters In	`DWORD*` must be a valid pointer. The value that it passes in must be between 1 `(NVPWRMZR_MIN_VALUE)` and 3 `(NVPWRMZR_MAX_VALUE)`.
Return Values	**`True`** if the PowerMizer level is obtained successfully. **`False`** for the following reasons: • The `DWORD*` pointer passed in is not valid. • The system does not support PowerMizer. • The hardware escape into the resource manager to obtain the PowerMizer level fails.