CHAPTER 1

Video Capture Device Drivers

Video capture device drivers provide low-level video capture services for Windows multimedia applications. Both applications and MCI device drivers can use these services to control video capture devices. These devices can provide services such as the following:

- Single frame video capture
- Real-time (streaming) video capture
- Video overlay
- Produce data in a standard or proprietary compressed format

Video capture devices must have a corresponding video capture device driver to be used with Windows. This chapter explains the Windows interface for video capture device drivers. It covers the following topics:

- The different types of video capture channels
- General information about writing a video capture device driver
- How a video capture device driver handles the system messages for the installable driver interface
- How a video capture device driver handles device specific messages for video capture
- An alphabetical reference to the messages and data structures used to write video capture device drivers

Before reading this chapter, you should be familiar with the video services available with Windows. You should also be familiar with the Windows installable driver interface. For information about the video services and the installable driver interface, see the *Microsoft Windows Programmer's Reference*. For information on other drivers using the installable driver interface, see the *Microsoft Windows Multimedia Device Adaptation Guide*.

Architecture of a Video Capture Driver

The MSVIDEO.DLL module provides the interface between client applications and video capture device drivers—applications do not call the drivers directly. When a client application calls a video capture function, MSVIDEO.DLL translates the call into a message and sends the message to the device driver.

Video Capture Device Driver Channels

Video capture device drivers can transfer data through four different channels. The destination or source of each channel is the frame buffer that is part of the video capture hardware. The four channels and the frame buffer are shown in the following illustration:

Data channels in the video capture driver.

The video capture channel (External In) is a source of video information placed in the frame buffer. The video source might be a video camera, video player, or television tuner. The format of both the incoming signal and the data placed in the frame buffer is controlled by the video capture hardware.

The video capture device can display the frame buffer data by using the video display channel (External Out). In practice, this could be with a second monitor or a video overlay device.

The device driver and application will use the video in channel (Video In) to transfer the video data to application supplied buffers.

The device driver and application can play captured data by using the video out channel (Video Out) to transfer data back into the frame buffer. Playback through this channel might be to review a sequence just captured or to play data from a file.

To supply minimum services, video capture drivers must support the External In and Video In channels. These channels provide services for video capture but not for video playback. Drivers with only External In and Video In channels rely on other system components (such as video compression and decompression drivers) for video playback.

Note: The Video Out channel is not currently used. The interface for video compression and decompression drivers is currently used to display this information.

The Video Capture Application

The application controlling the video capture driver is an integral part of the capture process. The application has the responsibility of allocating the memory used for video capture and managing the data buffers used for the transfer of data. If the user wants to capture audio simultaneously with video, the application also controls the audio driver used for capturing the input audio. Once the video and audio drivers capture the data, the application has the responsibility for any post-processing of this data. For example, if the application wants to save it as an AVI file, it must add the appropriate headers and create the AVI RIFF structure saved in the disk file.

Sample Device Drivers

The examples in this chapter were extracted from a sample device driver (BRAVADO.DRV) for the Truevision Bravado video capture hardware. The examples also apply to the Creative Labs Video Blaster (VBLASTER.DRV) capture hardware. The sample source code for this driver shares or parallels the sample source code for the

Bravado device driver. (The files that are unique for the two samples include the .H, .RC, .DEF, .DLL, .LIB, and MAKEFILEs.)

Like many of the newer frame grabbers, these devices use the PCVIDEO 9001 chipset from Chips and Technologies. The sample driver is designed to support any video capture device based on the PCVIDEO chipset. You can develop a device driver for this chipset in as little as a single day if the following assumptions are true:

- A DLL is available which is modeled after PCVIDEO.DLL from Chips and Technologies. Functions exported by the DLL may have different names, but they should have similar functionality. For example, Truevision supplies a DLL called VW.DLL. The sample driver, BRAVADO.DRV calls on the services of this DLL to access most low-level hardware functions.
- Internally, images are captured to memory using YUV 4:1:1 encoding.

Video capture devices that are not based on the PCVIDEO chipset, or that use alternate internal formats, will require additional work to develop routines to convert between formats and control the device. Devices which capture data with the RGB format can be readily supported by modifications to the sample code.

The Structure of a Video Capture Device Driver

Video capture device drivers are dynamic-link libraries (DLLs) usually written in C or assembly language, or a combination of the two languages. You should combine operations for different video capture channels in a single DLL. For example, the Bravado video capture driver module, BRAVADO.DRV, has operations for video capture as well as the display of live video using key color or overlay.

As installable drivers, these drivers will provide a **DriverProc** entry point used to process system messages. For general information about installable drivers, the **DriverProc** entry point, and system messages sent to this entry point, see the *Microsoft Windows Programmer's Reference*. This chapter includes supplemental information for the system messages. This information describes specifically how video capture drivers should respond to the system messages that are critical to their proper operation.

Video capture drivers also use the **DriverProc** entry point to process messages specifically for video capture. Information on how drivers use the **DriverProc** entry point to process these messages is contained in this chapter.

Combining Video Capture and Video Compression/Decompression Drivers

If the same hardware is required or used for a combination of video capture and video compression, you might combine both of these functions into a common DLL and use a single **DriverProc** entry point to service them. The common entry point will simplify the coordination of the different functions.

Note: Because video capture drivers can rely on video compression and decompression drivers for efficient operation, a single driver can handle both video capture, and video compression and decompression services. Video capture drivers use the VIDEO_OPEN_PARMS data structure when they are opened. This structure has the same field definitions as the ICOPEN structure used by video compression and decompression drivers. By examining the **fccType** field, a combined driver can determine whether it is being opened as a video capture driver or a video compression and decompression driver. (Video capture devices contain the four-character code 'vcap' in this field.) For more information on video compression and decompression drivers, see Chapter 10, “Video Compression and Decompression Drivers.”

Video Capture Header Files

The messages and data structures used exclusively by video capture device drivers are defined in MSVIDDRV.H. Functions, error returns, and constants used by both video capture device drivers and applications are defined in MSVIDEO.H

Naming Video Capture Device Drivers

The filenames for device driver DLLs are not required to have a file extension of “.DLL”—you can name your driver using any file extension you want. It is suggested that you use the extension “.DRV” for your device drivers to follow the convention set by Windows.

SYSTEM.INI Entries for Video Capture Device Drivers

The SYSTEM.INI file contains information for loading and configuring device drivers. Your device driver must be identified in the [drivers] section. Your device driver might also have entries in the [386enh] section if it requires any VxDs for operation. Your driver might also reserve a device-specific section in the SYSTEM.INI to store configuration information. For more information on this device-specific section, see “The Installable Driver Interface,” later in this chapter. The [drivers] and [386enh] sections are updated by an installation program when your device is installed or removed.

The preferred method for installing device drivers uses the Drivers option in the Control Panel. The Drivers option uses information in the OEMSETUP.INF file for your driver to add the entries in the [drivers] section as well as entries in the [386enh] section to install any VxDs you require. The procedures for creating an OEMSETUP.INF file are described in the Windows DDK.

The entry that identifies your driver in the [drivers] section lets Windows load the driver.

If this entry is absent, your driver won't be recognized. While installation programs normally add the necessary entry for completed device drivers, you might have to manually add it while you are developing your device driver. You might also have to manually add any [386enh] entries you need. The final version of your device driver should use an installation program to create and delete the entries in these two sections.

For video capture devices, a key name of “MSVideo” specifies the name of your driver in the [drivers] section of SYSTEM.INI. For example, the following extract identifies one video capture device driver named “BRAVADO.DRV”.

```
[drivers]
timer=timer.drv
joystick=ibmjoy.drv
MSVideo=bravado.drv
```

If there is more than one driver for a given device type, append a number from 1 to 9 after the key name. (When you have multiple drivers, use sequential numbers to identify them.)

While you can have more than one driver of the MSVIDEO type in the [drivers] section, the Drivers option in the Control Panel cannot install multiple drivers of this type. To work with more than one video driver, you might use the Drivers option to remove the existing driver and install an alternate, or you might manually edit SYSTEM.INI file to include the additional MSVIDEO entries. If you manually edit SYSTEM.INI, you can select the driver used when you execute the video capture application. The following example shows a [drivers] section with entries for five video capture drivers:

```
[drivers]
msvideo=targa16.drv
msvideo1=testdrv.drv
msvideo2=bravado.drv
msvideo3=vblaster.drv
msvideo4=MYDRVR.DRV
```

If you are using the VIDCAP video capture application, you can select the video capture driver it uses with the -d command line option. The integer specified after the -d corresponds to the video capture driver entry. For example, VIDCAP -d0 uses the TARGA16.DRV driver associated with the msvideo entry. VIDCAP -d3 uses the VBLASTER.DRV associated with the msvideo3 entry.

Note: Video capture device drivers are loaded only when needed by an application.

The Module-Definition File

To build a device-driver DLL, you must have a module-definition (.DEF) file. In this file, you must export the **DriverProc** entry-point function. Functions are exported by ordinal, as shown in the following example BRAVADO.DEF file:

```
LIBRARY     AVIBRAV

DESCRIPTION 'MSVIDEO:Truevision Bravado Driver'

EXETYPE     WINDOWS

PROTMODE

CODE       MOVEABLE  DISCARDABLE  LOADONCALL
DATA       FIXED  SINGLE  PRELOAD

SEGMENTS    _TEXT   FIXED               PRELOAD
        INIT    MOVEABLE   DISCARDABLE   PRELOAD
        VCAP    MOVEABLE   DISCARDABLE   PRELOAD

HEAPSIZE    1024

EXPORTS     WEP             @1  RESIDENTNAME
        DriverProc        @2  RESIDENTNAME
```

The actual ordinal values you assign to each exported function are not significant, though each must be unique within the DLL.

For more information on the entry-point function listed in this example, see “Entry-Point Function” later in this chapter.

The Module Name Line

The module name line should specify a unique module name for your device driver. Windows will not load two different modules with the same module name. It's a good idea to use the base of your driver filename since, in certain instances, **LoadLibrary** will assume that to be your module name.

The Module Description Line

The module description line in the module-definition file should specify the type of device the driver supports. For example, here's the module description line from the module-definition file for the Bravado video capture driver:

```
DESCRIPTION 'MSVIDEO: Truevision Bravado Driver'
```

Use MSVIDEO followed by a colon (:) to indicate the type of device your driver supports.

The Drivers option in the Control Panel uses these names to identify different types of drivers and to create the entry in the [Drivers] section of SYSTEM.INI when installing a driver.

Considerations for Interrupt-Driven Drivers

Most video capture device drivers will be interrupt-driven. For example, a video input device interrupts when the device receives a new video frame. Driver code accessed during an interrupt service routine must adhere to the guidelines discussed in the following sections.

Fixing Code and Data Segments

Any code segments or data segments a driver accesses at interrupt-time must be fixed segments. For best overall system performance, you should minimize the amount of code and data in fixed segments. To minimize the amount of fixed code, isolate all interrupt-time code in a few source modules and put this code into a single fixed code segment. Unless your driver has a large amount of data not accessed at interrupt time, use a single fixed data segment.

The Bravado video capture driver is a medium-model DLL, using a single data segment and multiple code segments. The following example fragment is from the module-definition file for the Bravado device driver:

```
CODE      MOVEABLE  DISCARDABLE  LOADONCALL
DATA      FIXED  SINGLE  PRELOAD

SEGMENTS    _TEXT   FIXED              PRELOAD
        INIT    MOVEABLE   DISCARDABLE   PRELOAD
        VCAP    MOVEABLE   DISCARDABLE   PRELOAD
```

This example fixes the data segment and the code segment named _TEXT. All other code segments are moveable.

The code segment _TEXT is used as a safety measure. The compiler places code for which you do not specify a segment in the _TEXT segment. This way any code that is missed will be placed into a fixed segment preventing possible problems at interrupt time. However, you should check your segmentation to ensure that only code that is required to be FIXED goes into the FIXED code segment.

Allocating and Using Memory

You can allocate either local memory or global memory for use at interrupt time.

To allocate local memory for use at interrupt time, follow these steps:

1. Use **LocalAlloc** with the LMEM_FIXED flag to get a handle to the memory block. (This assumes fixed data segments.)
2. Pass this handle to **LocalLock** to get a near pointer to the memory block.

Any global memory a driver uses at interrupt-time must be page-locked. To allocate and page-lock global memory, follow these steps:

3. Use **GlobalAlloc** with the GMEM_MOVEABLE and GMEM_SHARE flags to get a handle to the memory block.
4. Pass this handle to **GlobalLock** to get a far pointer to the memory block.
5. Pass the handle to **GlobalPageLock** to page-lock the memory block.

Calling Windows Functions at Interrupt Time

The only Windows functions a driver can call at interrupt time are **PostMessage**, **PostAppMessage**, **DriverCallback**, **timeGetSystemTime**, **timeGetTime**,

timeSetEvent, **timeKillEvent**, **midiOutShortMsg**, **midiOutLongMsg**, and **OutputDebugStr**.

The Installable Driver Interface

The entry-point function, **DriverProc**, processes messages sent by the system to the driver as the result of an application call to a low-level video capture function. For example, when an application opens a video capture device, the system sends the specified video capture device driver a DRV_OPEN message. The driver's **DriverProc** function receives and processes this message.

Note: Your driver should respond to all system messages. If supplemental information is not provided for them in this chapter, use the definitions provided in the *Microsoft Windows Programmer's Reference*.

An Example **DriverProc** Entry-Point Function

The video capture driver uses the **DriverProc** function for its entry-point. The following example is extracted from the Bravado video capture driver.

```
LRESULT FAR PASCAL _loadds DriverProc(DWORD dwDriverID, HDRVR hDriver,
UINT uiMessage, LPARAM lParam1, LPARAM lParam2)
{
switch (uiMessage)
   {

      case DRV_LOAD:
         return (LRESULT)1L; //Device loaded successfully


      case DRV_FREE:
         return (LRESULT)1L; //Device freed successfully

      case DRV_OPEN:
         // lParam2 is NULL when the user configures
         // the device driver with the Drivers Option of the
         // Control Panel. If opened without an open structure,
         // return a dummy (non-zero) ID so OpenDriver will work.
         if (lParam2 == NULL)
            return BOGUS_DRIVER_ID;


         // Verify this open is for a video capture driver, and
         // not for an installable compressor/decompressor
         if (((LPVIDEO_OPEN_PARMS) lParam2) -> fccType != OPEN_TYPE_VCAP)
            return 0L;


         return (DWORD)(WORD)
            VideoOpen ((LPVIDEO_OPEN_PARMS) lParam2);
```

```
case DRV_CLOSE:
        //Device opened without an open structure
        if (dwDriverID == BOGUS_DRIVER_ID || dwDriverID == 0)
           return 1L;      // Device closed

        //Close device if termination routine executed successfully
        return ((VideoClose((PCHANNEL)dwDriverID)
           == DV_ERR_OK) ? 1L : 0);


case DRV_ENABLE:
        // Enable the driver: initialize hardware, hook
        // interrupts, allocate DMA buffer, etc.
        return (LRESULT)1L;


case DRV_DISABLE:
        //Disable the driver: free DMA buffer, unhook
        //interrupts, reset hardware, etc.
        return (LRESULT)1L;


case DRV_QUERYCONFIGURE:
        return (LRESULT)1L;       // Driver supports configuration


case DRV_CONFIGURE:
        // The Drivers option of the Control Panel sends this
        // message to display a dialog box that lets the user configure
        // the driver. For example, set the port base and interrupt.
        return (LRESULT)Config((HWND)lParam1, ghModule);


case DRV_INSTALL:
        return (LRESULT)DRV_OK; //Driver installed OK


case DRV_REMOVE:
        // The driver is being removed from the installed drivers list.
        // The driver should remove its .INI section, etc.
        ConfigRemove();
        return (LRESULT)DRV_OK; //Driver removed OK

    .
    .
    .
```

```
        default:
            if (dwDriverID == BOGUS_DRIVER_ID || dwDriverID == 0)
                return DefDriverProc(dwDriverID, hDriver, uiMessage,
                                     lParam1, lParam2);


            // Process video capture driver specific messages
            return VideoProcessMessage((PCHANNEL)dwDriverID,
                                       uiMessage, lParam1, lParam2);
    }
}
```

Handling the DRV_OPEN and DRV_CLOSE Messages

Like other installable drivers, client applications must open a video capture device before using it and close it when finished using it, so the device will be available to other applications. When a driver receives an open request, it returns a value that the system will use for *dwDriverID* sent with subsequent messages. When your device driver receives other messages, it can use this value to identify instance data needed for operation. Drivers can use the instance data for information related to the client that opened a device.

It's up to you to decide if your device driver will support more than one client simultaneously. If you do this, though, remember to check the *dwDriverID* parameter to determine which channel is being accessed.

For DRV_OPEN, the *lParam2* parameter contains a pointer to a VIDEO_OPEN_PARMS data structure containing information about the open. This structure has the following fields:

```
typedef struct {
    DWORD   dwSize;
    FOURCC  fccType;
    FOURCC  fccComp;
    DWORD   dwVersion;
    DWORD   dwFlags;
    DWORD   dwError;
} VIDEO_OPEN_PARMS;
```

The **fccType** field of this structure will contain the four character code 'vcap'. Because of the four video capture channels, video capture drivers must examine the flags set in the **dwFlags** field of the VIDEO_OPEN_PARMS data structure to determine the type of channel being opened. Your driver should be prepared to open (and conversely, close) the video channels in any order.

The following flags are defined for the video channels:

VIDEO_EXTERNALIN

An external input channel responsible for loading images into the frame buffer.

VIDEO_EXTERNALOUT

An external output channel responsible for displaying images in the frame buffer to an external or system monitor, or to an overlay device.

VIDEO_IN

A video input channel responsible for transferring images from the frame buffer to system memory. This might include a translation step or reformatting of the image. For example, reformatting a 16-bit RGB image to an 8 bit palette image.

VIDEO_OUT

A video output channel responsible for transferring images into the frame buffer from the CPU. (The sample driver does not use this channel type.)

The **dwVersion** field specifies the version of the video capture command set used by MSVIDEO.DLL. The version number lets your driver identify the command set to determine its capabilities. For the initial release of the video capture command set, your driver does not have to detect and adjust itself for multiple versions of the command set. Future versions of your driver can use this value to enable new features that depend on new capabilities of the video capture command set.

The **dwSize** field specifies the size of the **VIDEO_OPEN_PARMS** structure.

The **fccComp** field is unused.

The **dwError** field specifies an error value the driver might return to the client application if it fails the open.

The following code fragment illustrates the routines the Bravado device driver uses to handle the DRV_OPEN and DRV_CLOSE messages. This device driver supports only one instance of each video channel.

```
PCHANNEL NEAR PASCAL VideoOpen( LPVIDEO_OPEN_PARMS lpOpenParms)
{
   PCHANNEL    pChannel;
   DEVICE_INIT di;
   LPDWORD     lpdwError = &lpOpenParms->dwError;
   DWORD       dwFlags = lpOpenParms-> dwFlags;

   *lpdwError = DV_ERR_OK;
```

```
// Initialize hardware on first call
if (!fDeviceInitialized) {

    // Get Port/IRQ/Base/etc. in INI file
    GetHardwareSettingsFromINI (&di);

    // Perform hardware initialization
    if (! HardwareInit (&di)) {
        *lpdwError = DV_ERR_NOTDETECTED;
        return NULL;
    }

    ConfigGetSettings();   // Get global hue, sat, channel, zoom

    .
    .
    .
    // Deleted code initializes hardware & sets remaining global values
    .
    .
    .
}


// Get instance memory. On exit this function assigns this value
// to dwDeviceID. By using this value for dwDeviceID,
// the device driver can easily retrieve the instance data
// when it needs it to process subsequent messages.
pChannel = (PCHANNEL)LocalAlloc (LPTR, sizeof(CHANNEL));
if (pChannel == NULL)
    return (PCHANNEL) NULL;


//  make sure the channel is not already in use
switch ( dwFlags &
      ( VIDEO_EXTERNALIN | VIDEO_EXTERNALOUT | VIDEO_IN | VIDEO_OUT) ) {


    case VIDEO_EXTERNALIN:
        if( gwCaptureUsage >= MAX_CAPTURE_CHANNELS)
            goto error;
        gwCaptureUsage++;
        break;


    case VIDEO_EXTERNALOUT:
        if( gwDisplayUsage >= MAX_DISPLAY_CHANNELS)
            goto error;
        gwDisplayUsage++;
        break;


    case VIDEO_IN:
        if( gwVideoInUsage >= MAX_IN_CHANNELS)
            goto error;
        gwVideoInUsage++;
        break;
```

```
        case VIDEO_OUT:
            if( gwVideoOutUsage >= MAX_OUT_CHANNELS)
                goto error;
            gwVideoOutUsage++;
            break;


        default:
            goto error;
    }


    //  Now that the hardware is allocated initialize instance structure

    pChannel->fccType          = OPEN_TYPE_VCAP;
    pChannel->dwOpenType       =
        (dwFlags & (VIDEO_EXTERNALIN|VIDEO_EXTERNALOUT|VIDEO_IN|VIDEO_OUT));
    pChannel->dwOpenFlags      = dwFlags;
    pChannel->lpVHdr           = NULL;
    pChannel->dwError          = 0L;


    gwDriverUsage++;
    return pChannel;


error:
    if (pChannel)
        LocalFree((HLOCAL)pChannel);


    *lpdwError = DV_ERR_ALLOCATED;
    return NULL;
}
```

The following example shows the function used to close the example video capture device driver:

```
DWORD NEAR PASCAL VideoClose(PCHANNEL pChannel)
{
    // Decrement the channel open counters


    switch (pChannel-> dwOpenType) {


        case VIDEO_EXTERNALIN:
            gwCaptureUsage--;
            break;


        case VIDEO_EXTERNALOUT:
            gwDisplayUsage--;
            break;
```

```
        case VIDEO_IN:
            // If started, or buffers in the queue,
            // return error and don't close
            if (gfVideoInStarted || lpVHdrFirst)
                return DV_ERR_STILLPLAYING;

            gwVideoInUsage--;
            break;


        case VIDEO_OUT:
            gwVideoOutUsage--;
            break;

        default:
            break;
    }


    gwDriverUsage--;  // Overall driver useage count

    if (gwDriverUsage == 0) {
        HardwareFini ();        // Shut down the device
        TransFini ();           // Free the translation table
        FreeFrameBufferSelector ();  // Free the frame buffer selector
        fDeviceInitialized = FALSE;
    }

    // Free the instance data
    LocalFree((HLOCAL)pChannel);

    return DV_ERR_OK;
}
```

Handling the DRV_ENABLE and DRV_DISABLE Messages

The example **DriverProc** function calls the functions **Enable** and **Disable** to do the work of enabling and disabling the driver. These functions are device dependent.

Generally, when a driver is enabled, you initialize the hardware, hook interrupts, allocate any memory that you need, and set a flag to indicate the driver is enabled. The exact sequence your device driver will follow is determined by the requirements and structure of your device driver. For example, the Bravado device driver uses interrupts only for streaming data. When enabled, it will hook its interrupts only if it was disabled while streaming data.

If your driver has not been enabled by MMSYSTEM, or if it failed the enable process, the driver should return MMSYSERR_NOTENABLED for any messages it receives from client applications. When a driver is disabled, you free any memory that you allocated, unhook interrupts, reset the hardware, and set a flag to indicate the driver is not enabled.

It's possible for a driver to receive a DRV_DISABLE message while it is in the process of capturing data. For example, this can happen when the user switches to a MS-DOS

application when Windows is running in standard mode. Video capture device drivers should behave as if the driver were stopped with a DVM_STREAM_STOP message and then restarted with a DVM_STREAM_START message when it receives a DRV_DISABLE/DRV_ENABLE message pair.

Driver Configuration

Installable drivers can supply a configuration dialog box for users to access through the Drivers option in the Control Panel. The Drivers option sends the DRV_CONFIGURE message to your driver to display the dialog box.

The dialog box should display the name and version number of your device driver. If your device driver supports different interrupt-level and port assignments, it should also support user configuration through the Drivers option in the Control Panel.

Interrupt-level and port assignments, and any other hardware-related settings, can be stored in a section with the same name as the driver in the user's SYSTEM.INI file. For example, the following SYSTEM.INI section created by the Bravado example driver specifies interrupt level 9 and memory base E:

```
[Bravado.drv]
Interrupt=9
MemoryBase=E
```

Alternatively, your driver might use its own INI file for this information.

Video Capture Driver Messages

This section gives the device driver specific messages for video capture device drivers. See “Video Capture Device Driver Reference,” later in this chapter, for detailed information on these messages.

Configuring the Channels of a Video Capture Driver

In addition to the configuration dialog box displayed for the DRV_CONFIGURE message, video capture drivers can display a dialog box for each channel. These dialog boxes are the primary means of setting parameters in your device driver. The following message requests that the device driver display a dialog box:

DVM_DIALOG

Displays a dialog box which controls a video channel.

When your device driver first gets this message, use the handle in *lParam1* to determine which channel is being configured. The Bravado example driver determines the channel

from the flags used to open it. It saves these flags as part of its instance data created when it was opened.

The dialog box displayed for each channel sets the characteristics for each channel. If a channel does not support configuration, return DV_ERR_NOTSUPPORTED. The following table suggests the contents of each dialog box:

Channel	Dialog Box Description
VIDEO_EXTERNALIN	Displays a dialog box which controls how video (either analog or digital) is captured. The dialog box might set attributes such as contrast and brightness.
VIDEO_EXTERNALOUT	Displays a dialog box which controls how video is displayed on a second monitor or video adapter such as a video overlay card.
VIDEO_IN	Displays a dialog box which controls how video is transferred from the frame buffer.
VIDEO_OUT	Displays a dialog box which controls how video is transferred to the frame buffer.

When processing the DVM_DIALOG message, check *lParam2* for the VIDEO_DLG_QUERY flag prior to displaying the dialog box. If an application uses this flag, it is only determining if a video channel supports a dialog box. For this flag, return DV_ERR_OK if the video channel supports a dialog box. If not, return DV_ERR_NOTSUPPORTED in response to the message.

The Bravado example driver uses the following function to handle the DVM_DIALOG message (this function is called from the Bravado VideoProcessMessage function):

```
DWORD NEAR PASCAL VideoDialog (DWORD dwOpenType, HWND hWndParent, DWORD dwFlags)
{

   switch (dwOpenType) {

      case VIDEO_EXTERNALIN:
         if (dwFlags & VIDEO_DLG_QUERY)
            return DV_ERR_OK;      // Channel has a dialog box
         DialogBox(ghModule, MAKEINTRESOURCE(DLG_VIDEOSOURCE),
               (HWND)hWndParent, VideoSourceDlgProc);
         break;


      case VIDEO_IN:
         if (dwFlags & VIDEO_DLG_QUERY)
            return DV_ERR_OK;      // Channel has a dialog box
         DialogBox(ghModule, MAKEINTRESOURCE(DLG_VIDEOFORMAT),
               (HWND)hWndParent, VideoFormatDlgProc);
         break;


      case VIDEO_OUT:
         return DV_ERR_NOTSUPPORTED; //Channel does not have a dialog box


      case VIDEO_EXTERNALOUT:
         if (dwFlags & VIDEO_DLG_QUERY)
            return DV_ERR_OK;      // Channel has a dialog box
         DialogBox(ghModule, MAKEINTRESOURCE (DLG_VIDEODISPLAY),
               (HWND)hWndParent, VideoMonitorDlgProc);
         break;

      default:
         return DV_ERR_NOTSUPPORTED;
   }
   return DV_ERR_OK;
}
```

Video capture drivers might save the settings from these dialog boxes in the section reserved for your device driver in the SYSTEM.INI file. Your driver should append this information to the entries created for the DVR_CONFIGURE messages to this section. For example, the example Bravado driver might have this section in the SYSTEM.INI file:

```
[Bravado.drv]
Interrupt=9
MemoryBase=E
Hue=10
Saturation=6
InputChannel=2
Contrast=24
```

Alternatively, a device driver might implement its own method of storing configuration information for each channel.

Setting and Obtaining Video Capture Format

The video capture format globally defines the attributes of the images transferred from the frame buffer with the video in channel. Attributes include image dimensions, color depth, and the compression format of images transferred. Applications use the following message to set or retrieve the format of the digitized image:

DVM_FORMAT

Assigns or obtains format information.

The calling application must modify this message with flags to indicate its purpose. Your driver must examine the flags sent with the message to determine the proper response. The flags are specified in *lParam1*. The following flags help define the meaning of the messages:

VIDEO_CONFIGURE_QUERY

Determines if the driver supports the message.

VIDEO_CONFIGURE_QUERYSIZE

Requests the size of the format data structure.

VIDEO_CONFIGURE_SET

Indicates values are being sent to the driver.

VIDEO_CONFIGURE_GET

Indicates the application is interrogating the driver.

The VIDEO_CONFIGURE_GET or VIDEO_CONFIGURE_SET flag indicates if the DVM_FORMAT message is being used to obtain the format or set the format. The DVM_FORMAT message and these flags are sent to your driver when it is opened, and when it is configured with DVM_DIALOG.

When an application opens your driver, it retrieves the initial driver format. (Video capture drivers initially default to a format that efficiently uses the capabilities of the video capture hardware or, if they have been previously configured, they restore the last user specified configuration saved in a disk file.) If this format is acceptable, the application continues its operations. If the format is not acceptable, the application will either immediately close your driver or suggest a very limited format. If the limited format is not acceptable to your driver, the application closes it. (Typically, applications do not accept a format because they cannot allocate enough memory to capture video. A limited format might free enough memory for operation.)

Applications also get the format when the user changes the format. (Users change the

format with the VIDEO_IN channel dialog box displayed with the DVM_DIALOG message.) In this case, applications get and retain a copy of the current format prior to sending the DVM_DIALOG message. After the user exits from the DVM_DIALOG dialog box, applications get the new format from your driver. If the application accepts the new format, it uses the VIDEO_CONFIGURE_SET flag to return the format back to your driver. (Your driver should verify that the application has not changed the format information.) If the application does not accept the new format, it restores the format it obtained prior to displaying the dialog box.

The DVM_FORMAT messages uses *lParam2* to pass the format information. This parameter contains a pointer to a VIDEOCONFIGPARMS structure. This structure has the following fields:

```
typedef struct tag_video_configure_parms {
   LPDWORD  lpdwReturn;
   LPVOID   lpData1;
   DWORD    dwSize1;
   LPVOID   lpData2;
   DWORD    dwSize2;
} VIDEOCONFIGPARMS;
```

The **lpData1** field points to a BITMAPINFOHEADER data structure. The size of this structure is specified in **dwSize1**.

Changing the format can affect overall dimensions of the active frame buffer as well as bit depth and color space representation. Since changing between NTSC and PAL video standards can also affect image dimensions, applications should request the current format following display of the EXTERNAL_IN channel dialog box.

If an application just wants to know if your driver supports DVM_FORMAT, it sends the VIDEO_CONFIGURE_QUERY flag with the message. (Using the VIDEO_CONFIGURE_QUERY flag without VIDEO_CONFIGURE_GET or VIDEO_CONFIGURE_SET is invalid.) Your device driver should return DV_ERR_OK if it supports the message. Otherwise, it should return DV_ERR_NOTSUPPORTED.

If an application wants to determine the amount of memory it needs to allocate for the format information, it sends the DVM_FORMAT message with the VIDEO_CONFIGURE_GET and VIDEO_CONFIGURE_QUERYSIZE flags set. Your driver should specify the format size in the **lpdwReturn** field of the VIDEOCONFIGUREPARMS structure.

Setting and Obtaining the Video Source and Destination Rectangles

Video capture drivers might support a source rectangle to specify a portion of an image that is digitized or transferred to the display. External in ports use the source rectangle to specify the portion of the analog image digitized. External out ports use the source rectangle to specify the portion of frame buffer shown on the external output.

Similarly, video capture drivers might support a destination rectangle to specify the

portion of the frame buffer or screen used to receive the image. External in ports can use a destination rectangle to specify the portion of the frame buffer used for the digitized video input. External out ports can use the rectangle to specify the client rectangle on the display.

The following messages are used to set and obtain the video source and destination rectangles:

DVM_DST_RECT

Sets and retrieves the destination rectangle used by video devices.

DVM_SRC_RECT

Sets and retrieves the source rectangle used by video devices.

The calling application must modify these messages with flags to indicate their exact meaning. Your driver must examine the flags sent with the messages to determine the proper response. The flags are specified in *lParam2*. The following flags define the meaning of the DVM_DST_RECT and DVM_SRC_RECT messages:

VIDEO_CONFIGURE_SET

Indicates values are being sent to the driver.

VIDEO_CONFIGURE_GET

Indicates the application is interrogating the driver.

VIDEO_CONFIGURE_QUERY

Determines if the driver supports the message.

The VIDEO_CONFIGURE_SET flag indicates the application is setting a source or destination rectangle. The rectangle coordinates are specified in a RECT structure pointed to by *lParam1*.

If an application sets a source or destination rectangle for an external out channel, your driver will normally receive a series of messages. For these channels, applications normally send both DVM_SRC_RECT and DVM_DST_RECT to your driver to properly set the rectangles. The application follows these messages with DVM_UPDATE. Video overlay devices should paint their key color in response to DVM_UPDATE.

Applications use VIDEO_CONFIGURE_GET to determine the coordinates of the source and destination rectangles. Applications use additional flags with VIDEO_CONFIGURE_GET to indicate if they want the coordinates of the rectangle currently defined, the maximum size of the rectangle, or the minimum size of the rectangle. The following flags are defined for these operations:

VIDEO_CONFIGURE_MIN

Used with VIDEO_CONFIGURE_GET to determine the minimum rectangle supported.

VIDEO_CONFIGURE_MAX

Used with VIDEO_CONFIGURE_GET to determine the maximum rectangle supported.

VIDEO_CONFIGURE_CURRENT

Used with VIDEO_CONFIGURE_GET to determine the current rectangle.

Your driver should return the coordinates for the appropriate rectangle in the RECT structure pointed to by *lParam1*.

An application uses VIDEO_CONFIGURE_QUERY to determine if your driver supports VIDEO_CONFIGURE_QUERY or VIDEO_CONFIGURE_SET. (The VIDEO_CONFIGURE_QUERY flag without VIDEO_CONFIGURE_GET or VIDEO_CONFIGURE_SET is invalid.) Your device driver should return DV_ERR_OK if it supports the flag. Otherwise, it should return DV_ERR_NOTSUPPORTED.

Determining Channel Capabilities

Channel capabilities include overlaying video, scaling of images with the source and destination rectangles, and clipping of images with the source and destination rectangles. The following message retrieves the channel capabilities of a driver:

DVM_GET_CHANNEL_CAPS

Return the capabilities of a channel to the application.

Applications use DVM_GET_CHANNEL_CAPS to obtain information about the capabilities of a channel. The *lParam1* parameter specifies a far pointer to a

CHANNEL_CAPS data structure and the *lParam2* parameter specifies its size. The **CHANNEL_CAPS** structure has the following fields:

```
typedef struct channel_caps_tag {
   DWORD  dwFlags;
   DWORD  dwSrcRectXMod;
   DWORD  dwSrcRectYMod;
   DWORD  dwSrcRectWidthMod;
   DWORD  dwSrcRectHeightMod;
   DWORD  dwDstRectXMod;
   DWORD  dwDstRectYMod;
   DWORD  dwDstRectWidthMod;
   DWORD  dwDstRectHeightMod;
} CHANNEL_CAPS;
```

Your driver should use the **dwFlags** field to return flags indicating its capabilities for overlaying video, and clipping and scaling images with the source and destination rectangles. The following flags are defined:

VCAPS_OVERLAY

Indicates the channel is capable of overlay. This flag is used only for **EXTERNAL_OUT** channels.

VCAPS_SRC_CAN_CLIP

Indicates that the source rectangle can be set smaller than the maximum dimensions.

VCAPS_DST_CAN_CLIP

Indicates that the destination rectangle can be set smaller than the maximum dimensions.

VCAPS_CAN_SCALE

Indicates that the source rectangle can be a different size than the destination rectangle.

If your driver supports changing the size and position of the source rectangle, it should indicate the finest granularity used for changes to the rectangle in the **dwSrcRectXMod**, **dwSrcRectYMod**, **dwSrcRectWidthMod**, and **dwSrcRectHeightMod** fields.

If your driver supports changing the size and position of the destination rectangle, it should indicate the finest granularity used for changes to the rectangle in the **dwDstRectXMod**, **dwDstRectYMod**, **dwDstRectWidthMod**, and **dwDstRectHeightMod** fields. If a channel supports arbitrarily positioned rectangles, with arbitrary sizes, the values above should all be set to 1.

Your driver returns DV_ERR_OK if the message was processed successfully. It returns DV_ERR_NOTSUPPORTED if the message is not supported.

Setting and Obtaining a Video Capture Palette

Applications can set and retrieve the palette used with captured video. This gives applications the ability to control and modify the palette used for video sequences. The palette messages apply only to the video in and video out channels. The following messages apply to the video capture palette:

DVM_PALETTE

Assigns or obtains palette information.

DVM_PALETTERGB555

Associates an RGB555 palette with a video device channel.

The calling application must modify these messages with flags to indicate their purpose. Your driver must examine the flags sent with the messages to determine the proper response. The flags are specified in *lParam1*. The following flags define the meaning of the messages:

VIDEO_CONFIGURE_SET

Indicates values are being sent to the driver.

VIDEO_CONFIGURE_GET

Indicates the application is interrogating the driver.

VIDEO_CONFIGURE_QUERY

Determines if the driver supports the message.

VIDEO_CONFIGURE_QUERYSIZE

Requests the size of the palette data structure.

The VIDEO_CONFIGURE_GET or VIDEO_CONFIGURE_SET flag modifies the DVM_PALETTE message to indicate that the driver should return the current palette or that the driver should set a new palette. The *lParam2* parameter used with DVM_PALETTE contains a pointer to a VIDEOCONFIGPARMS data structure. This structure has the following fields:

```
typedef struct tag_video_configure_parms {
   LPDWORD  lpdwReturn;
   LPVOID   lpData1;
   DWORD    dwSize1;
   LPVOID   lpData2;
   DWORD    dwSize2;
} VIDEOCONFIGPARMS;
```

If the VIDEO_CONFIGURE_SET flag is used with DVM_PALETTE, the **lpData1** field

points to a LOGPALETTE structure containing the new palette. The size of the memory allocated for the LOGPALETTE structure is specified in the **dwSize1** field.

If the VIDEO_CONFIGURE_GET flag is used with DVM_PALETTE, the **lpData1** field points to a LOGPALETTE structure used to retrieve the palette. The size of the memory allocated for the LOGPALETTE structure is specified in the **dwSize1** field. Your driver should transfer the palette to the structure indicated by **lpData1**.

If an application just wants to determine the size of the palette, it sends the DVM_PALETTE message with both the VIDEO_CONFIGURE_GET and VIDEO_CONFIGURE_QUERYSIZE flags. Your driver should return the palette size in the **lpdwReturn** field.

If an application just wants to know if your driver supports DVM_PALETTE and its flags, it also sets the VIDEO_CONFIGURE_QUERY flag with VIDEO_CONFIGURE_GET or VIDEO_CONFIGURE_SET. (The VIDEO_CONFIGURE_QUERY flag without VIDEO_CONFIGURE_GET or VIDEO_CONFIGURE_SET is invalid.) Your device driver should return DV_ERR_OK if it supports the DVM_PALETTE message and the operation indicated with the set or get flag. Otherwise, it should return DV_ERR_NOTSUPPORTED.

DVM_PALETTE does not use the **lpData2** and **dwSize2** fields.

Applications use the DVM_PALETTERGB555 message to associate an RGB555 palette with a video device channel. Only the VIDEO_CONFIGURE_SET and VIDEO_CONFIGURE_QUERY flags apply to this message. The VIDEO_CONFIGURE_SET flag modifies the DVM_PALETTERGB555 message to indicate that the driver should set a new palette. The *lParam2* parameter used with DVM_PALETTERGB555 contains a pointer to a VIDEOCONFIGPARMS data structure.

When setting the palette, the **lpData1** field points to a LOGPALETTE structure containing the new palette. The **lpData2** field points to a 32 kilobyte RGB555 translation table. The device driver uses this table to translate the RGB555 triples into palette colors when capturing data in an 8 bit palette mode. The **dwSize1** and **dwSize2** fields specify the size of the structures indicated by **lpData1** and **lpData2**.

If an application just wants to know if your driver supports DVM_PALETTERGB555, it sends the VIDEO_CONFIGURE_QUERY flag with VIDEO_CONFIGURE_SET. (The VIDEO_CONFIGURE_QUERY flag without VIDEO_CONFIGURE_SET is invalid.) Your device driver should return DV_ERR_OK if it supports the DVM_PALETTERGB555 message. Otherwise, it should return DV_ERR_NOTSUPPORTED.

The following example shows how the Bravado device driver handles the DVM_PALETTE message. The structure for DVM_PALETTERGB555 is similar.

```
DWORD NEAR PASCAL VideoConfigureMessage(PCHANNEL pChannel, UINT msg, LONG lParam1, LONG
lParam2)
{
   LPVIDEOCONFIGPARMS lpcp;
   LPDWORD     lpdwReturn;  // Return parameter from configure
   LPVOID      lpData1;     // Pointer to data1
   DWORD       dwSize1;     // size of data buffer1
   LPVOID      lpData2;     // Pointer to data2
   DWORD       dwSize2;     // size of data buffer2
   DWORD       dwFlags;


   if (pChannel-> dwOpenType != VIDEO_IN)
      return DV_ERR_NOTSUPPORTED;

   dwFlags = lParam1;

   lpcp = (LPVIDEOCONFIGPARMS) lParam2;
   lpdwReturn = lpcp-> lpdwReturn;
   lpData1 = lpcp-> lpData1;
   dwSize1 = lpcp-> dwSize1;
   lpData2 = lpcp-> lpData2;
   dwSize2 = lpcp-> dwSize2;


switch (msg) {

   case DVM_PALETTE:

      switch (dwFlags) {

         case (VIDEO_CONFIGURE_QUERY | VIDEO_CONFIGURE_SET):
         case (VIDEO_CONFIGURE_QUERY | VIDEO_CONFIGURE_GET):
            return DV_ERR_OK;

         case VIDEO_CONFIGURE_QUERYSIZE:
         case (VIDEO_CONFIGURE_QUERYSIZE | VIDEO_CONFIGURE_GET):
           *lpdwReturn = sizeof(LOGPALETTE) +
               (palCurrent.palNumEntries-1) *
               sizeof(PALETTEENTRY);
           break;


         case VIDEO_CONFIGURE_SET:
         case (VIDEO_CONFIGURE_SET | VIDEO_CONFIGURE_CURRENT):
            if (!lpData1)    // points to a LOGPALETTE structure.
               return DV_ERR_PARAM1;
            return (SetDestPalette ( (LPLOGPALETTE) lpData1,
              (LPBYTE) NULL));
            break;


         case VIDEO_CONFIGURE_GET:
         case (VIDEO_CONFIGURE_GET | VIDEO_CONFIGURE_CURRENT):
           return (GetDestPalette ( (LPLOGPALETTE) lpData1,
                  (WORD) dwSize1));
           break;
```

```
            default:
              return DV_ERR_NOTSUPPORTED;

          } // end of DVM_PALETTE switch

          return DV_ERR_OK;
          .
          .
          .


        default:        // Not a message that we understand
          return DV_ERR_NOTSUPPORTED;

        } // end of message switch

        return DV_ERR_NOTSUPPORTED;
}
```

Obtaining the Device Driver Version

The following message lets an application interrogate your device driver to determine the version of the video capture command set.

DVM_GETVIDEOAPIVER

Obtains the version of the video capture command set.

Your driver should return VIDEOAPIVERSION in the DWORD buffer that *lParam1* points to. This message does not have any flags associated with it.

Transferring Data From the Frame Buffer

The following message lets an application obtain a single frame from the frame buffer:

DVM_FRAME

Obtains a single frame from the frame buffer.

This message is the basis for the simplest form of video capture. Applications might use this to record animated sequences created frame-by-frame or to capture a single still image such as a photograph. The following sequence of operations occurs when a client application requests the transfer of a single video frame:

1. The client allocates the memory for the data buffer.
2. The client sets a pointer to the empty data buffer in the VIDEOHDR data structure.

3. The client sends the device driver a pointer to the VIDEOHDR data structure with the **videoFrame** function. (The destination channel must be a VIDEO_IN channel.)
4. When the device driver receives the DVM_FRAME messages that Windows sends in response to **videoFrame**, it fills the data buffer with information from the frame buffer and updates the VIDEOHDR data structure. Note that the buffer might not have been prepared.
5. When the device driver has filled a data buffer, it returns from the DVM_FRAME message. This returns control back to the client.
6. After the client has finished with the data, it frees the memory used for the data.

Streaming Video Capture

Video capture device drivers use the DVM_STREAM messages sent to a VIDEO_IN channel to stream full motion video to the client application. Your device driver will use the following messages while it is streaming video:

DVM_STREAM_INIT

Initializes a video input stream.

DVM_STREAM_PREPAREHEADER

Requests that the driver prepare a data buffer for input.

DVM_STREAM_ADDBUFFER

Adds a buffer to the video input stream queue.

DVM_STREAM_START

Begins streaming video input.

DVM_STREAM_STOP

Ends video input streaming.

DVM_STREAM_UNPREPAREHEADER

Requests that a driver clean up the preparation previously done on a data buffer.

DVM_STREAM_FINI

Closes and deallocates a video stream.

The Data Transfer Model For Streaming Video Input

The data transfer model for streaming video input is similar to the model defined for the waveform device drivers. If you have worked with the waveform device drivers, many of the concepts used there will be usable with video capture device drivers.

The following sequence of operations occurs when streaming video data between a video capture device driver and a client application:

1. The client allocates the memory buffers for the video data.
2. The client initializes the data stream (DVM_STREAM_INIT).
3. The client requests that the driver prepare the data buffers (DVM_STREAM_PREPAREHEADER).
4. The client sends the empty data buffers to the driver (DVM_STREAM_ADDBUFFER).
5. The driver puts the data buffers in its input queue.
6. When the streaming operation begins with DVM_STREAM_START, the driver fills a data buffer and sets the done bit for the data buffer. The driver will then release the buffer from its queue and proceed to fill the next buffer.
7. When the client is ready for data, it uses the done bit or callback to see if the data in the buffer is ready.
8. After the client empties the buffer, it resets the done bit and sends the empty buffer back to the driver for it to add to its queue (DVM_STREAM_ADDBUFFER).

Once the stream starts, the client application and the video capture driver do not communicate directly. The video capture driver fills the data buffers at the rate specified by the client application using the frame rate information provided with the DVM_STREAM_INIT message. It fills the buffers without waiting for any synchronization signal from the application as long as buffers are available and it is not paused or stopped by the application. The buffers are filled in the order that the driver receives them from the application. (If the device driver runs out of buffers, it should set an error flag. A client application can use the DVM_STREAM_GETERROR message to test for this condition.)

The client application expects the buffers back in the order that it sends them to the device driver. When it is ready for more data, it will check the done bit of the next buffer it expects to receive from the device driver. If the done bit is set, the application continues operation with that buffer. If the done bit is not set, the application will periodically check the done bit while it waits for the buffer.

Streaming continues until it is stopped by the application. The following sequence of operations occurs when the application has finished capturing data:

- When the client stops the streaming operation with DVM_STREAM_STOP, the driver stops filling buffers.
- If the client wants to restart streaming, it sends DVM_STREAM_START. If the client is finished streaming, it requests that the driver unprepare the data buffers (DVM_STREAM_UNPREPAREHEADER).
- The client releases the data stream (DVM_STREAM_FINI) and frees the memory allocated for the video data.

Initializing the Data Stream

The DVM_STREAM_INIT message initializes a video device for data streaming. This message must precede all other streaming messages for a channel.

The *lParam1* parameter of DVM_STREAM_INIT specifies a far pointer to a VIDEO_STREAM_INIT_PARMS structure and the *lParam2* specifies its size in bytes. The VIDEO_STREAM_INIT_PARMS structure has the following fields:

```
typedef struct tag_video_stream_init_parms {
   DWORD  dwMicroSecPerFrame;
   DWORD  dwCallback;
   DWORD  dwCallbackInst;
   DWORD  dwFlags;
   DWORD  hVideo;
} VIDEO_STREAM_INIT_PARMS;
```

The different channels handle the message and data structure in different ways.

For external in channels, DVM_STREAM_INIT enables capture of images into the frame buffer. External in channels should expect this message at any time. They can ignore the **dwMicroSecPerFrame**, **dwCallback**, and **dwCallbackInst** fields. The **dwFlags** field must contain the VIDEO_ALLOWSYNC flag for synchronous devices.

For video in channels, DVM_STREAM_INIT sets the capture rate and callback information. The **dwMicroSecPerFrame** field specifies the number of microseconds between successive capture frames. The **dwCallback** field contains the address of a callback function or the handle to a window called during video streaming. (This parameter is set to NULL if a callback function or window is not used.) The callback procedure processes any messages related to the progress of recording. If a callback function address is specified, **dwFlags** is set to CALLBACK_FUNCTION. If the application has any data to pass to the callback function, it specifies the data in **dwCallbackInst**. If a callback window handle is specified, **dwFlags** is set to CALLBACK_WINDOW. Drivers can also use **DriverCallback** to send a message to a window or callback function. For more information on **DriverCallback**, see the *Windows Multimedia Device Adaptation Guide.* For more information on using the video capture callback, see the “Video Capture Device Driver Reference.”

For external out channels, DVM_STREAM_INIT enables overlay display. External out channels should expect this message at any time. They can ignore the

dwMicroSecPerFrame, **dwCallback**, and **dwCallbackInst** fields. The **dwFlags** field contains any flags that might affect the external out channel.

All channels return DV_ERR_OK if the message was processed successfully. All channels should return DV_ERR_ALLOCATED if the channel is already allocated or DV_ERR_NOMEM if they are unable to allocate or lock memory.

Preparing Data Buffers

Because video data buffers must be accessed at interrupt time, the memory allocated for them is subject to the requirements mentioned previously in "Considerations for Interrupt-Driven Drivers." Rather than have the client application prepare the memory before sending data blocks to the driver, the client requests that the driver do the preparation.

Most drivers can respond to the DVM_STREAM_PREPAREHEADER and DVM_STREAM_UNPREPAREHEADER messages) by returning a DV_ERR_UNSUPPORTED error. When your driver returns DV_ERR_UNSUPPORTED, the system will perform the necessary preparation on the data block. This consists of page locking the header and data sections so the driver can access them at interrupt time.

If your device driver does not need the data to be page locked (for example, if you immediately copy the data to an on-card buffer) or if you have additional preparation to do to the buffer, you might respond to these messages yourself instead of having the system handle them. You should respond to both DVM_STREAM_PREPAREHEADER and DVM_STREAM_UNPREPAREHEADER, or to neither.

Starting and Stopping Streaming

DVM_STREAM_START starts a video stream. For video in channels, this message begins transferring the contents of the frame buffer to the system supplied buffers. In response to DVM_STREAM_START, your driver should enable the interrupts it needs and begin capturing the images and copying them to the application supplied buffers.

DVM_STREAM_STOP stops a video stream. When a video in channel receives this message, it stops filling buffers and retains any empty buffers remaining in the queue. Your driver can disable any interrupts it needs to capture data, however, it should be prepared to receive the DVM_STREAM_START message to resume capturing data. If data capture has not started, this message has no effect and the device driver returns DV_ERR_OK.

Neither DVM_STREAM_START nor DVM_STEAM_STOP use *lParam1* or *lParam2*. Your driver should return DV_ERR_OK if it processed the message successfully. It should return DV_ERR_NOTSUPPORTED if it does not support the message.

Ending Capture

The DVM_STREAM_FINI message terminates data streaming. This should always be the last streaming message received by a channel.

For external in channels, DVM_STREAM_FINI disables capture of images into the frame buffer. External in channels should expect this message at any time. DVM_STREAM_FINI might not have a corresponding DVM_STREAM_INIT message.

For video in channels, DVM_STREAM_INIT finishes data streaming process. Your driver can free any resources that it used to capture data.

For external out channels, DVM_STREAM_INIT disables overlay display. External out channels should expect this message at any time. DVM_STREAM_FINI might not have a corresponding DVM_STREAM_INIT message.

All channels return DV_ERR_OK if the message was processed successfully. The video in channels should return DV_ERR_STILLPLAYING if there are still buffers in its queue.

Additional Stream Messages

The following messages are used in support of data streaming:

DVM_STREAM_RESET

Stops video input streaming and returns all data buffers to the client application.

DVM_STREAM_GETERROR

Returns the error encountered while streaming data.

DVM_STREAM_GETPOSITION

Requests the current position in the video stream.

The client application uses DVM_STREAM_RESET to stop data streaming and release all buffers. When your driver gets this message it should return to the state set with DVM_STREAM_INIT.

The client application uses DVM_STREAM_GETERROR to obtain the error status of a channel. The *lParam1* and *lParam2* parameters point to two DWORDS your driver should use to return error information. Fill the DWORD specified by *lParam1* with the value of the most recent error. Typically, the error encountered is DV_ERR_NO_BUFFERS. If your driver has not encountered an error or if it receives this message when a stream is not initialized, set the DWORD to DV_ERR_OK. Fill the DWORD specified by *lParam2* with the number of frames dropped because of the error.

After processing this message your driver should reset its error value and count of frames dropped. Drivers that do not have access to interrupts might use this message to trigger buffer processing.

Return DV_ERR_OK if your driver processes the message without an error. If your driver does not support this message, return DV_ERR_NOTSUPPORTED.

Applications use the DVM_STREAM_GETPOSITION message to retrieve the current position of the video in stream. The *lParam1* parameter specifies a far pointer to a **MMTIME** data structure and the *lParam2* parameter specifies its size. The **MMTIME** structure has the following fields:

```
typedef struct mmtime_tag {
   UINT  wType;
   union {
      DWORD  ms;
      DWORD  sample;
      DWORD  cb;
      struct {
         BYTE  hour;
         BYTE  min;
         BYTE  sec;
         BYTE  frame;
         BYTE  fps;
         BYTE  dummy;
      } smpte;
      struct {
         DWORD  songptrpos;
      } midi;
   } u;
} MMTIME;
```

When your device gets DVM_STREAM_POSITION, it should check the **wtype** field. If your driver does not support the format specified, it specifies its current time format in the field. The application checks the format specified in this field when the message returns.

Video capture drivers typically return time in the millisecond format. Normally, your driver sets the position to zero when streaming starts with DVM_STREAM_START.

Your driver should returns DV_ERR_OK if it processed the message successfully. It can return DV_ERR_PARM1 if the data structure supplied for the format has invalid data or DV_ERR_SIZEFIELD if the data structure is too small.

Video Capture Device Driver Reference

This section is an alphabetic reference to the messages and data structures provided by Windows for use by video capture device drivers. There are separate sections for messages and data structures. The messages and data structures are defined in MSVIDDRV.H and MSVIDEO.H.

Video Capture Device Driver Message Reference

Windows communicates with video capture device drivers through messages sent to the driver. The driver processes these messages with its **DriverProc** entry-point function.

This section contains an alphabetical list of the video capture messages that can be received and sent by video capture device drivers. Each message name contains a prefix, identifying the type of the message.

A message consists of three parts: a message number and two DWORD parameters.

Message numbers are identified by predefined message names. The two DWORD parameters contain message-dependent values.

Message Summary

The following messages are used for error handling:

DVM_GETERRORTEXT

This message retrieves a string which contains the description of an error.

DVM_STREAM_GETERROR

This message returns the last error encountered by a channel.

The following messages are used for configuring the device driver and obtaining information from it:

DVM_DIALOG

This message displays a dialog box which controls video parameters for a channel.

DVM_DST_RECT

This message sets and retrieves the destination rectangle used by a video device channel.

DVM_FORMAT

This message is for configuring the format of the video device channel.

DVM_GET_CHANNEL_CAPS

This message is used to return the capabilities of a channel to the application.

DVM_GETVIDEOAPIVER

This message returns the version of the video API used by the driver.

DVM_PALETTE

This message sets and retrieves a logical palette used by a video device channel.

DVM_PALETTERGB555

This message associates an RGB555 palette with a video device channel.

DVM_SRC_RECT

This message sets and retrieves the source rectangle used by a video device channel.

The following messages are used for capturing data:

DVM_FRAME

This message processes a single frame from the video device.

DVM_STREAM_ADDBUFFER

This message sends an input buffer to a video device.

DVM_STREAM_FINI

This message terminates streaming on a video channel.

DVM_STREAM_GETPOSITION

This message retrieves the current position of the stream.

DVM_STREAM_INIT

This message initializes a video device for streaming.

DVM_STREAM_PREPAREHEADER

This message prepares an input buffer for video streaming.

DVM_STREAM_RESET

This message stops input of a video stream and resets the current position to 0.

DVM_STREAM_START

This message starts a video stream.

DVM_STREAM_STOP

This message stops a video stream.

DVM_STREAM_UNPREPAREHEADER

This message cleans up the preparation performed by **DVM_STREAM_PREPAREHEADER**.

DVM_UPDATE

This message is used with a **EXTERNAL_OUT** channel to indicate that the display needs to be updated.

The following messages are used with video callback functions:

MM_DRVM_CLOSE

This message is sent to a video callback function or window when a video channel is closed.

MM_DRVM_DATA

This message is sent to a video callback function or window when the specified buffer is being returned to the application.

MM_DRVM_ERROR

This message is sent to a video callback function or window when an error has occurred.

MM_DRVM_OPEN

This message is sent to a video callback function or window when a video channel is opened.

Video Capture Device Driver Messages

This section contains an alphabetical list of the video capture messages that can be received and sent by video capture device drivers. Each message name contains a prefix, identifying the type of the message.

A message consists of three parts: a message number and two DWORD parameters. Message numbers are identified by predefined message names. The two DWORD parameters contain message-dependent values.

DVM_DIALOG

This message displays a dialog box for setting the video parameters of a channel.

Parameters

DWORD *dwParam1*

Specifies the handle to the parent window.

DWORD *dwFlags*

Specifies flags for the dialog box. The following flag is defined:

VIDEO_DLG_QUERY

If this flag is set, the driver immediately returns DV_ERR_OK if it supplies a dialog box for the channel, or DV_ERR_NOTSUPPORTED if it does not.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following errors are defined:

DV_ERR_INVALHANDLE

Specified device handle is invalid.

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments Typically, this dialog box lets the user configure a video channel. For example, a VIDEO_IN channel might supply a dialog box to let the user select image dimensions and bit depth. Each channel type (VIDEO_IN, VIDEO_OUT, VIDEO_EXTERNALIN, and VIDEO_EXTERNALOUT) can have a unique configuration dialog box.

DVM_DST_RECT

This message sets and retrieves the destination rectangle used by a video device channel.

Parameters LPRECT lpDstRect

A far pointer to a **RECT** structure.

DWORD *dwFlags*

Specifies flags that indicate the type of transfer requested. Either the VIDEO_CONFIGURE_SET or the VIDEO_CONFIGURE_GET flag must be set, specifying the direction of the transfer. The following flags are defined:

VIDEO_CONFIGURE_SET
Send a rectangle to the device driver.

VIDEO_CONFIGURE_GET
Get the current rectangle from the device driver.

VIDEO_CONFIGURE_MIN
Get the minimum destination rectangle from the device driver.

VIDEO_CONFIGURE_MAX
Get the maximum destination rectangle from the device driver.

VIDEO_CONFIGURE_CURRENT
Get or set the current destination rectangle.

VIDEO_CONFIGURE_QUERY
This flag is used to query the device driver to determine if it supports the message.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments

The use of the destination rectangle for a channel depends on the channel type. For the VIDEO_EXTERNALIN channel, the destination rectangle specifies the location in the frame buffer used to digitize the image. This rectangle is specified in pixel coordinates.

For the VIDEO_EXTERNALOUT channel, the destination rectangle specifies the location used to display the overlay image. This rectangle is given in Windows screen coordinates.

For the VIDEO_IN and VIDEO_OUT channels, the destination rectangle is currently undefined.

DVM_FORMAT

This message is used for configuring the format of the VIDEO_IN channel.

Parameters

DWORD *dwFlags*

Specifies flags to indicate the type of format transfer requested. Either the VIDEO_CONFIGURE_SET or the VIDEO_CONFIGURE_GET flag must be set, specifying the direction of the transfer. The following flags are defined:

VIDEO_CONFIGURE_SET
Set the current format.

VIDEO_CONFIGURE_GET
Get the current format.

VIDEO_CONFIGURE_QUERY
Queries the driver whether it supports the message.

VIDEO_CONFIGURE_QUERYSIZE
Returns the size in bytes of the format in **lpdwReturn**. This flag must be used with VIDEO_CONFIGURE_GET.

LPVIDEOCONFIGPARMS *lpVConfigParms*

Specifies a far pointer to a **VIDEOCONFIGPARMS** structure. This structure has the following fields:

lpdwReturn
Specifies a far pointer to a **DWORD**. If the **VIDEO_CONFIGURE_QUERYSIZE** flag is used, the driver fills this field with the size (in bytes) of the **BITMAPINFOHEADER** data structure.

lpData1

Specifies a far pointer to a **BITMAPINFOHEADER** data structure.

dwSize1

Specifies the size in bytes of the **BITMAPINFOHEADER** data structure.

lpData2

Not used.

dwSize2

Not used.

Return Value

Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments

The **DVM_FORMAT** message globally defines the attributes of the frame buffer. This includes dimensions, color depth, and compression of images transferred with **DVM_FRAME** and buffers transferred during streaming capture. Changing the format may affect overall dimensions of the active frame buffer as well as bit depth and color space representation. Since changing between NTSC and PAL video standards can also affect image dimensions, applications should request the current format following display of the VIDEO_EXTERNALIN channel dialog box.

DVM_FRAME

This message transfers a single frame from the video device.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **VIDEOHDR** structure identifying the buffer.

DWORD *dwParam2*

Contains the size of the **VIDEOHDR** structure.

Return Value

Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_SIZEFIELD

Specified field size is too small.

Comments

This message returns immediately after transferring the frame. For a VIDEO_IN channel, this message transfers an image from the hardware frame buffer to the buffer specified in the **VIDEOHDR**. For a VIDEO_OUT channel, this message transfers an image from the buffer specified in the **VIDEOHDR** to the hardware frame buffer.

DVM_GET_CHANNEL_CAPS

This message is used to return the capabilities of a channel to the application.

Parameters

LPCHANNEL_CAPS *lpChannelCaps*

Specifies a far pointer to a **CHANNEL_CAPS** data structure.

DWORD *dwSize*

Specifies the size of the **CHANNEL_CAPS** data structure.

Return Value

Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

DVM_GETERRORTEXT

This message retrieves a string describing an error.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **VIDEO_GETERRORTEXT_PARMS** structure. The structure identifies the error number and return buffer.

DWORD *dwParam2*

Not used.

Return Value

Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_BADERRNUM

Indicates the specified error number is out of range.

Comments

If the error description is longer than the specified buffer, the description is truncated. The returned error string is always null-terminated. If the size of the return buffer is zero, a string description is not returned and DV_ERR_OK is used as the return value.

DVM_GETVIDEOAPIVER

This message returns the version of the video capture command set used by the driver.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a DWORD which will be filled with the version.

DWORD *dwParam2*

Not used.

Return Value Returns DV_ERR_OK if the message is successful.

DVM_PALETTE

This message sets and retrieves a logical palette used by a video device channel. This message applies only to VIDEO_IN and VIDEO_OUT channels.

Parameters DWORD *dwFlags*

Specifies any flags that indicate the type of palette transfer requested. Either the VIDEO_CONFIGURE_SET or the VIDEO_CONFIGURE_GET flag must be set, specifying the direction of the transfer. The following flags are defined:

VIDEO_CONFIGURE_SET
Send a palette to the driver.

VIDEO_CONFIGURE_GET
Get the current palette from the driver.

VIDEO_CONFIGURE_QUERY
This flag is used to query the driver to determine if it supports the message.

VIDEO_CONFIGURE_QUERYSIZE
Returns the size in bytes of the palette in **lpdwReturn**. This flag is only valid if the VIDEO_CONFIGURE_GET flag is also set.

LPVIDEOCONFIGPARMS *lpVConfigParms*

A far pointer to a **VIDEOCONFIGPARMS** structure. The **VIDEOCONFIGPARMS** structure has the following fields:

lpdwReturn is a far pointer to a **DWORD**. If the VIDEO_CONFIGURE_QUERYSIZE flag is used, the driver fills this field with the size of the logical palette (in bytes).

lpData1 is a far pointer to a **LOGPALETTE** structure.

dwSize1 is the size in bytes of the **LOGPALETTE**.

lpData2 is not used.

dwSize2 is not used.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments A palette is used when converting between frame buffer internal data formats and 8-bit palettized DIBs.

See Also DVM_PALETTERGB

DVM_PALETTERGB555

This message associates an RGB555 palette with a video device channel. Applications can provide an RGB555 translation table to a driver for fast conversions between RGB formats and 8 bit palettized formats. This message applies only to VIDEO_IN and VIDEO_OUT channels.

Parameters DWORD *dwFlags*

Specifies the flags indicating the type of palette transfer requested. The following flags are defined:

VIDEO_CONFIGURE_SET
Indicates values are being sent to the driver.

VIDEO_CONFIGURE_QUERY
This flag, when combined with VIDEO_CONFIGURE_SET is used to query the driver to determine if it supports the message.

LPVIDEOCONFIGPARMS *lpVConfigParms*

Specifies a far pointer to a **VIDEOCONFIGPARMS** data structure. This data structure has the following fields:

lpdwReturn
Not used.

lpData1
Specifies a far pointer to a **LOGPALETTE** data structure.

dwSize1
Specifies the size (in bytes) of the **LOGPALETTE** data structure.

lpData2
Specifies a far pointer to a 32 kilobyte RGB555 translation table. This table is used by the device driver to translate from RGB555 triplets into palette colors when capturing in 8 bit palette mode.

dwSize2
Specifies the size of the translate table in bytes. This value must be 32,768.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following errors are defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

DV_ERR_CREATEPALETTE

The device driver was not able to associate the palette with the video device channel.

DV_ERR_PARM1

The information supplied for *dwParam1* is invalid.

DV_ERR_PARM2

The information supplied for *dwParam2* is invalid.

DV_ERR_SIZEFIELD

The data structure supplied for the format is too small.

Comments

A translation table provides a fast method of converting between RGB and palettized color spaces. The palette index corresponding to an RGB color is found by indexing the translation table at xRRRRRGGGGGBBBBB (the five most significant bits of each color component is used to create the index).

DVM_SRC_RECT

This message sets and retrieves the source rectangle used by a video device channel.

Parameters

LPRECT lpSrcRect

A far pointer to a **RECT** structure.

DWORD *dwFlags*

Specifies flags that indicate the type of transfer requested. Either the VIDEO_CONFIGURE_SET or the VIDEO_CONFIGURE_GET flag must be set, specifying the direction of the transfer. The following flags are defined:

VIDEO_CONFIGURE_SET
Send a source rectangle to the device driver.

VIDEO_CONFIGURE_GET
Get the current source rectangle from the device driver.

VIDEO_CONFIGURE_MIN
Get the minimum source rectangle from the device driver.

VIDEO_CONFIGURE_MAX
Get the maximum source rectangle from the device driver.

VIDEO_CONFIGURE_CURRENT

Get or set the current source rectangle.

VIDEO_CONFIGURE_QUERY

This flag is used to query the driver to determine if it supports the message.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments The use of the source rectangle for a channel depends on the channel type. For the VIDEO_EXTERNALOUT channel, the source rectangle specifies the portion of the frame buffer displayed in the overlay window, in pixel coordinates. For the VIDEO_IN, VIDEO_EXTERNALIN, and VIDEO_OUT channels, the source rectangle is currently undefined.

DVM_STREAM_ADDBUFFER

This message sends an input buffer to a video device. When the buffer is filled, the device sends it back to the application.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **VIDEOHDR** structure identifying the buffer.

DWORD *dwParam2*

Specifies the size of the **VIDEOHDR** structure.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following errors are defined:

DV_ERR_NONSPECIFIC

A buffer is not specified.

DV_ERR_UNPREPARED

The buffer was not prepared.

Comments The data buffer must be prepared with **DVM_STREAM_PREPAREHEADER** before it is passed with **DVM_STREAM_ADDBUFFER**. The **VIDEOHDR** data structure and the data buffer pointed to by its **lpData** field must be allocated with **GlobalAlloc** using the GMEM_MOVEABLE and GMEM_SHARE flags, and locked with **GlobalLock**.

DVM_STREAM_FINI

This message terminates streaming on a video channel. This should always be the last

streaming message received by a channel.

Parameters

DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value

Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following errors are defined:

DV_ERR_STILLPLAYING

There are still buffers in the queue.

Comments

If all the input buffers sent with **DVM_STREAM_ADDBUFFER** haven't been returned to the application, your driver should fail the message. Client applications should send **DVM_STREAM_RESET** to mark all pending buffers as done prior to sending **DVM_STREAM_FINI**.

For VIDEO_EXTERNALIN channels, this message halts capturing of data to the frame buffer.

For VIDEO_EXTERNALOUT channels that support overlay, this message disables the overlay video.

See Also

videoStreamInit

DVM_STREAM_GETERROR

This message returns the error status of a channel.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a DWORD that the device will fill with the value of the most recent error.

DWORD *dwParam2*

Specifies a far pointer to a DWORD that the device will fill with the number of frames dropped.

Return Value

Returns DV_ERR_OK if there is no error. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments

A device driver should reset its internal error values and count of frames dropped to zero after it processes this message.

Client applications should send this message frequently during capture since some device drivers that do not have access to interrupts use this message to trigger buffer processing.

DVM_STREAM_GETPOSITION

This message retrieves the current position of the VIDEO_IN stream.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **MMTIME** structure.

DWORD *dwParam2*

Specifies the size in bytes of the **MMTIME** structure.

Return Value

Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following errors are defined:

DV_ERR_PARM1

The data structure supplied for the format has invalid data.

DV_ERR_SIZEFIELD

The data structure supplied for the format is too small.

Comments

If a device does not support the format specified in the **wtype** field of the **MMTIME** data structure it specifies the current time format in the field. The application checks the format specified in this field when the message returns. Video capture drivers typically return time in the milliseconds format.

The device sets the position to zero when it receives the **DVM_STREAM_START** message.

DVM_STREAM_INIT

This message initializes a video device for streaming. This message must precede all other streaming messages for a channel.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **VIDEO_STREAM_INIT_PARMS** structure. This structure has the following fields:

dwMicroSecPerFrame
Contains the number of microseconds between successive capture frames.

dwCallback
Specifies the address of a callback function or the handle to a window

called during video streaming to process messages related to the progress of recording. This parameter can be NULL.

dwCallbackInst

Specifies the instance data passed to the callback function. This parameter is not used with window callbacks.

dwFlags

Specifies flags for opening the device. The following flags are defined:

CALLBACK_WINDOW

If this flag is specified, **dwCallback** is a window handle.

CALLBACK_FUNCTION

If this flag is specified, **dwCallback** is a callback function address.

DWORD *dwParam2*

Specifies the size, in bytes, of the data structure.

Return Value

Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following errors are defined:

DV_ERR_ALLOCATED

Specified resource is already allocated.

DV_ERR_NOMEM

Unable to allocate or lock memory.

Comments

If a window or callback function will receive callback messages, the device driver uses the following messages to indicate the progress of video input: **MM_DRVM_OPEN**, **MM_DRVM_CLOSE**, **MM_DRVM_DATA**, and **MM_DRVM_ERROR**.

If a callback function is used, it must reside in a DLL. You do not have to use **MakeProcInstance** to get a procedure-instance address for the callback function.

For VIDEO_EXTERNALIN channels, DVM_STREAM_INIT triggers capturing of data to the frame buffer.

For VIDEO_EXTERNALOUT channels with overlay capabilities, DVM_STREAM_INIT enables the overlay.

DVM_STREAM_PREPAREHEADER

This message prepares an input buffer for video streaming.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **VIDEOHDR** structure identifying the buffer.

DWORD *dwParam2*

Specifies the size of the **VIDEOHDR** structure.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following errors are defined:

DV_ERR_NOMEM

Unable to allocate or lock memory.

DV_ERR_NOTSUPPORTED

Unable to prepare data block. (This return lets MSVIDEO prepare the data block.)

Comments The **VIDEOHDR** data structure and the data block pointed to by its **lpData** field must be allocated with **GlobalAlloc** using the GMEM_MOVEABLE and GMEM_SHARE flags, and locked with **GlobalLock**. Preparing a header previously prepared will have no effect, and the message will return zero. Typically, this operation is used to ensure that the buffer will be available for use at interrupt time.

DVM_STREAM_RESET

This message stops input of a video stream and resets the current position to 0. All pending buffers are marked as done and returned to the application.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments When a device driver receives this message, it should return to the state established for **DVM_STREAM_INIT**.

DVM_STREAM_START

This message starts a video stream.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments For VIDEO_IN channels, this message begins transferring the contents of the frame buffer to the system supplied buffers.

DVM_STREAM_STOP

This message stops a video stream.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments When a device receives this message, it marks the current buffer as done and retains any empty buffers remaining in the queue. For the buffer marked as done, the device places the actual length of the data in the **dwBytesUsed** field of the **VIDEOHDR** structure.

If the input is not started, this message has no effect and the device driver returns DV_ERR_OK.

DVM_STREAM_UNPREPAREHEADER

This message cleans up the preparation performed by **DVM_STREAM_PREPAREHEADER**.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **VIDEOHDR** structure identifying the buffer.

DWORD *dwParam2*

Specifies the size of the **VIDEOHDR** structure.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following errors are defined:

DV_ERR_STILLPLAYING

The data buffer is still in the device driver's available queue.

DV_ERR_NOTSUPPORTED

Unable to handle data block preparation. (This return lets MSVIDEO unprepare the data block.)

Comments This message is the complementary message to **DVM_STREAM_PREPAREHEADER**. This message unlocks the data buffer. Unpreparing a buffer not previously prepared has no effect, and the device driver returns DRV_ERR_OK.

DVM_UPDATE

This message is used with a VIDEO_EXTERNALOUT channel to indicate the display needs updating. This is typically sent to an overlay device whenever its client window is moved, sized, or requires painting.

Parameters HWND hWndClient

Specifies a window handle to the client window in which the VIDEO_EXTERNALOUT channel is displayed.

HDC *hDC*

Specifies the device context to be repainted.

Return Value Returns DV_ERR_OK if the message was successful. Otherwise, it returns an error number. The following error is defined:

DV_ERR_NOTSUPPORTED

Message is not supported.

Comments This message is sent to a driver when video overlay is enabled and the overlay key color might need updating. Painting the key color is the responsibility of the driver. An application initiates this message whenever it receives a **WM_PAINT**, **WM_MOVE**, **WM_POSITIONCHANGED**, or **WM_SIZE** message.

This message always follows **DVM_SRC_RECT** and **DVM_DST_RECT** messages for the VIDEO_EXTERNALOUT channel.

The **DVM_STREAM_INIT** and **DVM_STREAM_FINI** messages are used to enable and disable the overlay.

MM_DRVM_CLOSE

This message is sent by a driver to a video callback function or window when a DVM_STREAM_FINI message is received.

Parameters

DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Comments

This message is used by video capture drivers, installable compressors, and other types of installable drivers whenever a device is closed.

MM_DRVM_DATA

This message is sent by a driver to a video callback function or window when the specified buffer is returned to the application.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **VIDEOHDR** structure identifying the buffer.

DWORD *dwParam2*

Not used.

Comments

For VIDEO_IN channels, buffers are returned when they have been filled. For VIDEO_OUT channels, buffers are returned after they are displayed. All buffers are returned for the DVM_STREAM_RESET message.

This message is used by video capture drivers, installable compressors, and other types of installable drivers to signal an application that new data is available.

MM_DRVM_ERROR

This message is sent by a device driver to a video callback function or window when an error has occurred.

Parameters

DWORD *dwParam1*

Specifies the error ID.

DWORD *dwParam2*

Not used.

Comments

Although a device driver can send this message for any reason, it most often indicates that no more buffers are available for video streaming.

The **MM_DRVM_ERROR** message is used by video capture drivers, installable compressors, and other types of installable drivers to signal an application that an error occurred.

MM_DRVM_OPEN

This message is sent by a driver to a video callback function or window when a DVM_STREAM_INIT message is received.

Parameters

DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Comments

This message is used by video capture drivers, installable video compressors, and other types of installable drivers whenever a device is opened.

Video Capture Device Driver Data Structure Reference

This section lists data structures used by video capture device drivers for Windows. The data structures are presented in alphabetical order. The structure definition is given, followed by a description of each field.

CHANNEL_CAPS

The **CHANNEL_CAPS** structure is used with the DVM_GET_CHANNEL_CAPS message to return the capabilities of a channel to an application.

```
typedef struct channel_caps_tag {
    DWORD  dwFlags;
    DWORD  dwSrcRectXMod;
    DWORD  dwSrcRectYMod;
    DWORD  dwSrcRectWidthMod;
    DWORD  dwSrcRectHeightMod;
    DWORD  dwDstRectXMod;
    DWORD  dwDstRectYMod;
    DWORD  dwDstRectWidthMod;
    DWORD  dwDstRectHeightMod;
} CHANNEL_CAPS;
```

Fields

The **CHANNEL_CAPS** structure has the following fields:

dwFlags

Returns flags giving information about the channel. The following flags are defined:

VCAPS_OVERLAY
Indicates the channel is capable of overlay. This flag is used only for VIDEO_EXTERNALOUT channels.

VCAPS_SRC_CAN_CLIP
Indicates that the source rectangle can be set smaller than the maximum dimensions.

VCAPS_DST_CAN_CLIP
Indicates that the destination rectangle can be set smaller than the maximum dimensions.

VCAPS_CAN_SCALE
Indicates that the source rectangle can be a different size than the destination rectangle.

dwSrcRectXMod

Returns the granularity allowed when positioning the source rectangle in the horizontal direction.

dwSrcRectYMod

Returns the granularity allowed when positioning the source rectangle in the vertical direction.

dwSrcRectWidthMod

Returns the granularity allowed when setting the width of the source rectangle.

dwSrcRectHeightMod

Returns the granularity allowed when setting the height of the source rectangle.

dwDstRectXMod

Returns the granularity allowed when positioning the destination rectangle in the horizontal direction.

dwDstRectYMod

Returns the granularity allowed when positioning the destination rectangle in the vertical direction.

dwDstRectWidthMod

Returns the granularity allowed when setting the width of the destination rectangle.

dwDstRectHeightMod

Returns the granularity allowed when setting the height of the source rectangle.

Comments

Some channels can only use source and destination rectangles which fall on 2, 4, or 8 pixel boundaries. Similarly, some channels only accept capture rectangles widths and heights that are multiples of a fixed value. Rectangle dimensions indicated by modulus operators are considered advisory. When requesting a particular rectangle, the application must always check the return value to insure the request was accepted by the driver. For example, if **dwDstRectWidthMod** is set to 64, the application might try to set destination rectangles with widths of 64, 128, 192, 256, ..., and 640 pixels. The driver might actually support a subset of these sizes and indicates the supported sizes with the return value of the **DVM_DST_RECT** message. If a channel supports arbitrarily positioned rectangles, with arbitrary sizes, the values above should all be set to 1.

VIDEO_GETERRORTEXT_PARMS

The **VIDEO_GETERRORTEXT_PARMS** structure specifies a return buffer for the error text.

```
typedef struct tag_video_geterrortext_parms {
   DWORD  dwError;
   LPSTR  lpText;
   DWORD  dwLength;
} VIDEO_GETERRORTEXT_PARMS;
```

Fields

The **VIDEO_GETERRORTEXT_PARMS** structure has the following fields:

dwError

Specifies the error number.

lpText

Specifies a far pointer to the error return buffer.

dwLength

Specifies the length of the error return buffer.

VIDEO_OPEN_PARMS

The **VIDEO_OPEN_PARMS** structure defines the type of channel to open on a video capture device.

```
typedef struct {
   DWORD   dwSize;
   FOURCC  fccType;
   FOURCC  fccComp;
   DWORD   dwVersion;
   DWORD   dwFlags;
   DWORD   dwError;
} VIDEO_OPEN_PARMS;
```

Fields

The **VIDEO_OPEN_PARMS** structure has the following fields:

dwSize

Specifies the size of the **VIDEO_OPEN_PARMS** structure.

fccType

Specifies a four-character code identifying the type of channel being opened. For capture devices, this is set to "vcap".

fccComp

Unused.

dwVersion

Specifies the current version number of the video capture command set in MSVIDEO.DLL.

dwFlags

Specifies flags used to indicate the type of channel. The following flags are defined:

VIDEO_EXTERNALIN
: Specifies a channel that loads data from an external source into a frame buffer. This can also be called the capture channel.

VIDEO_IN
: Specifies a channel that transfers data from the frame buffer to system memory.

VIDEO_OUT
: Specifies a channel that transfers data from system memory to the frame buffer.

VIDEO_EXTERNALOUT
: Specifies a channel that controls display of frame buffer images. Display might be either on a second monitor, or via overlay.

dwError

Specifies an error value the driver should return to the client application if it fails the open.

Comments

This structure is identical to the **IC_OPEN** structure used by installable compressors.

This lets a driver handle both video capture and decompressor messages with a single **DriverProc** entry point.

VIDEO_STREAM_INIT_PARMS

The **VIDEO_STREAM_INIT_PARMS** structure contains the fields used to initialize a video stream for video capture.

```
typedef struct tag_video_stream_init_parms {
   DWORD  dwMicroSecPerFrame;
   DWORD  dwCallback;
   DWORD  dwCallbackInst;
   DWORD  dwFlags;
   DWORD  hVideo;
} VIDEO_STREAM_INIT_PARMS;
```

Fields

The **VIDEO_STREAM_INIT_PARMS** structure has the following fields:

dwMicroSecPerFrame

Specifies the number of microseconds between the start of one frame capture and the start of the next.

dwCallback

An optional parameter which specifies an address to a callback function or a handle to a window called during video recording. The callback function or window processes messages related to the progress of recording.

dwCallbackInst

Specifies user instance data passed to the callback function. This parameter is not used with window callbacks.

dwFlags

Specifies flags for the data capture. The following flags are defined:

VIDEO_ALLOWSYNC
: If this flag is not specified, the device will fail to open if it is a synchronous device.

CALLBACK_WINDOW
: If this flag is specified, *dwCallback* contains a window handle.

CALLBACK_FUNCTION
: If this flag is specified, *dwCallback* contains a callback function address.

hVideo

Specifies a handle to the video channel.

VIDEOCONFIGPARMS

The **VIDEOCONFIGPARMS** structure is used to send or return message specific configuration parameters.

```
typedef struct {
   LPDWORD  lpdwReturn;
   LPVOID   lpData1;
   DWORD    dwSize1;
   LPVOID   lpData2;
   DWORD    dwSize2;
} VIDEOCONFIGPARMS;
```

Fields

The **VIDEOCONFIGPARMS** structure has the following fields:

lpdwReturn

Specifies a far pointer to a **DWORD** to be filled with a message specific return value.

lpData1

Specifies a far pointer to message-specific data.

dwSize1

Specifies the size in bytes of data passed in **lpData1**.

lpData2

Specifies a far pointer to message specific data.

dwSize2

Specifies the size in bytes of data passed in **lpData2**.

See Also

DVM_FORMAT, DVM_PALETTE, DVM_PALETTERGB555

VIDEOHDR

The **VIDEOHDR** structure defines the header used to identify a video data buffer.

```
typedef struct videohdr_tag {
   LPSTR  lpData;
   DWORD  dwBufferLength;
   DWORD  dwBytesUsed;
   DWORD  dwTimeCaptured;
   DWORD  dwUser;
   DWORD  dwFlags;
   DWORD  dwReserved[4];
} VIDEOHDR;
```

Fields

The **VIDEOHDR** structure has the following fields:

lpData

Specifies a far pointer to the video data buffer.

dwBufferLength

Specifies the length of the data buffer.

dwBytesUsed

Specifies the number of bytes used in the data buffer.

dwTimeCaptured

Specifies the time (in milliseconds) when the frame was captured relative to the first frame in the stream.

dwUser

Specifies 32 bits of user data.

dwFlags

Specifies flags giving information about the data buffer. The following flags are defined for this field:

VHDR_DONE
: Set by the device driver to indicate it is finished with the data buffer and it is returning the buffer to the application.

VHDR_PREPARED
: Set by Windows to indicate the data buffer has been prepared with **videoStreamPrepareHeader**.

VHDR_INQUEUE
: Set by Windows to indicate the data buffer is queued for playback.

VHDR_KEYFRAME
: Set by the device driver to indicate a key frame.

dwReserved[4]

Reserved for use by the device driver. Typically, these maintain a linked list of buffers in the queue.