CHAPTER 1

Video Compression and Decompression Drivers

Video compression and decompression drivers provide low-level video compression and decompression services for Microsoft Video for Windows. The compression and decompression algorithms used can be hardware or software based. This chapter explains the Windows interface for these drivers. It covers the following topics: XE "Video capture functions:low-level video capture services"§

- General information about writing a video compression and decompression driver
- Information on how a video compression and decompression driver handles the system messages for the installable driver interface
- Information on how a video compression and decompression driver handles messages specific to compressing and decompressing video data
- An alphabetical reference to the messages and data structures used to write video compression and decompression 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 these Windows topics, see the *Microsoft Windows Programmer's Reference*.

Architecture of a Video Compression and Decompression Driver

The following two block diagrams show the architecture of a video compression and decompression driver. While the diagrams show separate compression and decompression drivers, an actual driver usually combines both functions. The following illustration shows the architecture of a decompression driver:

Architecture for a decompression driver.

The following illustration shows a similar architecture for the compression driver:

Architecture for a compression driver.

The decompression driver and compression driver blocks represent your compression and decompression driver. The client-application block represents the system and application software that uses the services of your compression and decompression driver. Application software will always use the system software to access compression and decompression drivers.

The source of information used for decompression is represented by the AVI file block. Other sources of images can be used in place of this block. AVI files are RIFF files that contain audio and video data. The client-application maintains the RIFF format when it reads and writes the file. (Your driver will send and receive video data. The client-application will add and remove the RIFF tags.)

Compression drivers receive uncompressed data from the video source. Typically the video source is a disk file but it could also come from other video sources such as a video capture device. The video data can be either still bitmaps or motion video frames.

While a previous block diagram showed the decompression driver returning the uncompressed video to the client-application, your driver can have the capability to write directly to the display or display driver. These devices can replace a Windows video driver or work in conjunction with it. The following illustration shows a decompression driver with the ability to write to the video display:

Architecture for a decompression-video driver.

These drivers handle a set of messages, the ICM_DRAW messages, in addition to the decompression messages defined for the services that return the decompressed video to the client-application.

The ICSAMPLE Example Driver

The examples in this chapter were extracted from the ICSAMPLE example driver. This sample illustrates the interface between Windows and video compression and decompression drivers. The sample compresses data by extracting every tenth pixel from the source and discarding the other nine. It decompresses by replacing the nine missing pixels with their retained neighbor. XE "Video capture drivers:Targa+ sample driver"§ XE "Sample applications:Targa+ video capture driver"§ XE "Targa+ video capture sample driver"§

The Structure of a Video Compression and Decompression Driver

Video compression and decompression drivers are dynamic-link libraries (DLLs) usually written in C or assembly language, or a combination of the two languages.

As installable drivers, these drivers will provide a **DriverProc** entry point. 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 compression and decompression drivers should respond to the system messages that are critical to their proper operation.

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

Video Compression and Decompression Header Files

The messages and data structures used exclusively by video compression and decompression drivers are defined in COMPDDK.H.

Naming Video Compression and Decompression Drivers

The filenames for 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 drivers to follow the convention set by Windows. XE "Video capture drivers:writing drivers:file extension"§ XE "DLLs:file extension"§

SYSTEM.INI Entries for Video Compression and Decompression Drivers

The SYSTEM.INI file contains information for loading and configuring drivers. Your driver must be identified in the [Drivers] or [Installable Compressors] section. This entry lets Windows load the driver. If an entry for your driver is absent, it won't be recognized. While installation applications normally add the necessary entry for completed drivers, you might have to manually add it while you develop your driver. The final version of your driver should use an installation application to create and delete the entries in these two sections.

Identify your driver in the [Drivers] section if you want to use the Drivers option of the Control Panel to install or configure it. (This is the recommended method of installation.) The Drivers option obtains the information it needs to install the driver from an OEMSETUP.INF file you create for your driver. This file should be included on the distribution disk for your driver. For information about the files needed to install your driver, see the *Microsoft Windows Device Driver Adaptation Guide* and *Microsoft Windows Programmer's Reference*. For information on the Drivers option, see the *Microsoft Windows Programmer's Reference*.

Identify your driver in the [Installable Compressors] section if you want to use a custom installation application. If you use a custom application, it should update the [Installable Compressors] section when your driver is installed or removed.

Video compression and decompression drivers are identified by a key name of "VIDC." followed by its four-character code identifier. For example, the following [Installable Compressors] section of SYSTEM.INI identifies one video compression and decompression driver:

```
[Installable Compressors]
VIDC.SAMP = ICSAMPLE.DRV
```

SAMP is the four-character code identifier of the compressor. This driver has a file name of "ICSAMPLE.DRV".

The four-character code identifier must be unique. If you want to create a new four-character code identifier, register it with Microsoft to set up a standard definition and avoid any conflicts with other codes that might be defined. To register a code for a compression and decompression driver, request a *Multimedia Developer Registration Kit* from the following group:

Microsoft Corporation
Multimedia Systems Group
Product Marketing
One Microsoft Way
Redmond, WA 98052-6399

For more information on four character codes, see the *Microsoft Windows Multimedia Programmer's Guide*, *Microsoft Windows Multimedia Programmer's Reference*, and Chapter 4, "AVI Files."

For more information on storing configuration information in the SYSTEM.INI file, see "The Installable Driver Interface," later in this chapter.

The Module-Definition File

To build a 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 ICSAMPLE.DEF file:

```
LIBRARY    ICSAMPLE

DESCRIPTION 'VIDC.SAMP:Sample Decompression Driver'

STUB              'WINSTUB.EXE'
EXETYPE    WINDOWS

CODE       MOVEABLE  DISCARDABLE  LOADONCALL
DATA       MOVEABLE  SINGLE  PRELOAD

SEGMENTS       _TEXT   DISCARDABLE   PRELOAD

HEAPSIZE   128

EXPORTS
    WEP
    DriverProc
```

If you are using the Drivers option of the Control Panel, include the key name of VIDC, a period (.), and the four-character code for your driver in the DESCRIPTION entry. (Use a colon (:) to separate this entry from the text description.) The Drivers option uses this description when it lists the driver in the [Drivers] section of the SYSTEM.INI file. For example, the previous description for ICSAMPLE.DRV uses VIDC.SAMP: in the DESCRIPTION line. If you are using a custom installation application, you do not need to include this description information.

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

The Module Name Line

The module name line should specify a unique module name for your 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. For example, the previous fragment used LIBRARY ICSAMPLE.

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 video compression and decompression function. For example, when an application opens a video compression and decompression driver, the system sends the specified driver a DRV_OPEN message. The driver’s **DriverProc** function receives and processes this message. Your **DriverProc** should return ICERR_UNSUPPORTED for any messages that it does not handle.

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

A video compression and decompression driver uses the **DriverProc** function for its entry-point. The following example is derived from the ICSAMPLE driver:

```
LRESULT CALLBACK _loadds DriverProc(DWORD dwDriverID, HDRVR hDriver,
UINT uiMessage, LPARAM lParam1, LPARAM lParam2)
{
   INSTINFO *pi = (INSTINFO *)(UINT)dwDriverID;

   switch (uiMessage)
   {


     case DRV_LOAD:
         return (LRESULT) Load();

     case DRV_FREE:
         Free();
         return (LRESULT)1L;

     case DRV_OPEN:
         // If being opened without an open structure, return a non-zero
         // value without actually opening.
         if (lParam2 == 0L)
             return 0xFFFF0000;


         return (LRESULT)(DWORD)(WORD)Open((ICOPEN FAR *) lParam2);

     case DRV_CLOSE:
         if (pi)
           Close(pi);

         return (LRESULT)1L;

  /*****************************************************************************
     system configuration messages
  *****************************************************************************/

     case DRV_QUERYCONFIGURE:   // For configuration with Drivers option.
         return (LRESULT)0L;

     case DRV_CONFIGURE:
         return DRV_OK;


 /*****************************************************************************
     device specific messages
  *****************************************************************************/

       .
       .
       .


 /*****************************************************************************
     standard driver messages
  *****************************************************************************/
```

```
        case DRV_DISABLE:
        case DRV_ENABLE:
            return (LRESULT)1L;

        case DRV_INSTALL:
        case DRV_REMOVE:
            return (LRESULT)DRV_OK;
    }

    if (uiMessage << DRV_USER)
        return DefDriverProc(dwDriverID, hDriver,
            uiMessage,lParam1,lParam2);
    else
        return ICERR_UNSUPPORTED;
}
```

Handling the DRV_OPEN and DRV_CLOSE Messages

Like other installable drivers, client applications must open a video compression and decompression driver before using it, and they must close it when finished using it so the driver 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 driver receives other messages, it can use this value to identify instance data needed for operation. Your drivers can use this data to maintain information related to the client that opened the driver.

Compression and decompression drivers should support more than one client simultaneously. If you do this, though, remember to check the *dwDriverID* parameter to determine which client is being accessed. XE "DRV_OPEN message"§XE "Video capture drivers:writing drivers:multiple clients"§

If the driver is opened for configuration by the Drivers option of the Control Panel, *lParam2* contains zero. When opened this way, your driver should respond to the DRV_CONFIGURE and DRV_QUERYCONFIGURE messages.

If opened for compression or decompression services, *lParam2* contains a far pointer to an ICOPEN data structure. The ICOPEN data structure has the following fields:

```
typedef struct {
    DWORD  fccType;
    DWORD  fccHandler;
    DWORD  dwVersion;
    DWORD  dwFlags;
} ICOPEN;
```

The **fccType** field specifies a four-character code representing the type of stream being compressed or decompressed. For video streams, this will be 'vidc'.

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 it is opened. This structure has the same

field definitions as the ICOPEN structure. 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 capture drivers, see Chapter 11, “Video Capture Device Drivers.”

Other drivers that require close coordination with video compression and decompression drivers can also be combined with video compression and decompression drivers if they use a similar interface.

The **fccHandler** field specifies a four-character code identifying a specific compressor. The client-application obtains the four-character code from the entry in the SYSTEM.INI file used to open your driver. Your driver should not fail the open if it does not recognize the four-character code.

The **dwVersion** field specifies the version of the compressor interface used to open the driver. Your driver can use this information to determine the capabilities of the system software when future versions of it are available.

The **dwFlags** field contains a constant indicating the function of the driver. The following constants are defined:

ICMODE_COMPRESS

The driver is opened to compress data.

ICMODE_DECOMPRESS

The driver is opened to decompress data.

ICMODE_DRAW

The device driver is opened to decompress data directly to hardware.

ICMODE_QUERY

The driver is opened for informational purposes, rather than for actual compression.

The ICMODE_COMPRESS, ICMODE_DECOMPRESS, and ICMODE_DRAW flags indicate your driver is opened to compress or decompress data. Depending on the flag, your driver should prepare to handle ICM_COMPRESS, ICM_DECOMPRESS, or ICM_DRAW messages. Your driver should also prepare to handle all messages used to configure and interrogate your driver.

The ICMODE_QUERY flag indicates your driver is opened to obtain information. It should prepare to handle the ICM_ABOUT, ICM_GETINFO, and ICM_GETDEFAULTQUALITY messages.

Compressor Configuration

Video compression and decompression drivers can receive a series of configuration messages. System configuration messages are typically sent by the Drivers option of the Control Panel to configure the hardware. Video compression and decompression specific configuration messages are typically initiated by the client-application or from dialog boxes displayed by your driver. Your driver should use these messages to configure the driver.

Configuration Messages Sent by the System

The following system messages are used by video compression and decompression drivers for hardware configuration:

DRV_QUERYCONFIGURE

This system message is sent to determine if the driver supports configuration.

DRV_CONFIGURE

This Control Panel message is sent to let the driver display a custom configuration dialog box for hardware configuration.

Installable drivers can supply a configuration dialog box for users to access through the Drivers option in the Control Panel. If your driver supports different options, it should allow user configuration. Any hardware-related settings should be stored in a section with the same name as the driver in the user's SYSTEM.INI file.

Like other installable drivers, your driver will receive DRV_QUERYCONFIGURE and DRV_CONFIGURE messages from the Drivers option of the Control Panel. If your driver controls hardware that needs to be configured, it should return a non-zero value for the DRV_QUERYCONFIGURE system message and display a hardware configuration dialog box for the DRV_CONFIGURE system message.

Messages for Configuring the Driver State

The video compression and decompression specific configuration messages are typically initiated by the client-application or from dialog boxes displayed by your driver. Your driver should use these messages to configure the driver. The following messages apply specifically to video compression and decompression drivers:

ICM_CONFIGURE

This message displays a custom configuration dialog box for driver configuration.

ICM_GETSTATE

This message obtains the current driver configuration.

ICM_SETSTATE

This message sets the state of the compressor.

If your driver is configurable, it should support the ICM_CONFIGURE message for driver configuration. In addition, it should also use this message to set parameters for compression or decompression. Any options the user selects in the dialog box displayed for ICM_CONFIGURE should be saved as part of the state information referenced by the ICM_GETSTATE and ICM_SETSTATE messages.

The ICM_GETSTATE and ICM_SETSTATE messages query and set the internal state of your compression or decompression driver. State information is device dependent and your driver must define its own data structure for it. While the client-application reserves a memory block for the information, it will obtain the size needed for the memory block from your driver. If your driver receives ICM_GETSTATE with a NULL pointer for *dwParam1*, the client-application is requesting that your driver return the size of its state information. Conversely, if your driver receives ICM_GETSTATE with *dwParam1* pointing to a block of memory, and *dwParam2* specifying the size of the memory block, the client-application is requesting that your driver transfer the state information to the memory block.

When your driver receives ICM_SETSTATE with *dwParam1* pointing to a block of memory, and *dwParam2* specifying the size of the memory block, the client-application is requesting that your driver restore its configuration from the state information contained in the memory block. Before setting the state, your driver should verify the state information applies to your driver. One technique for verifying the data is to reserve the first DWORD in the state data structure for the four-character code used to identify your driver. If you set this DWORD for data returned for ICM_GETSTATE, you can use it to verify the data supplied with ICM_SETSTATE. If ICM_SETSTATE has a NULL pointer for *dwParam1*, it indicates that your driver should return to its default state.

State information should not contain any data that is absolutely required for data decompression—any such data should be part of the format you return for the ICM_DECOMPRESS_GET_FORMAT message. For information on the ICM_DECOMPRESS_GET_FORMAT message, see “Decompressing Video Data” later in this chapter.

Messages Used to Interrogate the Driver

The client-application uses the following messages to obtain or display information about your driver:

ICM_ABOUT

This message displays the about dialog box for the driver.

ICM_GETINFO

This message obtains information about the driver.

The client-application sends the ICM_ABOUT message to display your driver's about box. If the client-application sets *dwParam1* to -1, it wants to know if your driver supports display of an about box. Your driver returns ICERR_OK if it does, and it returns ICERR_UNSUPPORTED if it does not. Your driver should only display an about box if the client-application specifies a window handle in *dwParam1*. The window handle indicates the parent of the dialog box.

The client-application uses the ICM_GETINFO message to obtain a description of your driver. Your driver should respond to this message by filling in the ICINFO structure it receives with the message. The flags your driver sets in the structure tells the client-application what capabilities the driver supports. The ICINFO structure has the following fields:

```
typedef struct {
    DWORD  dwSize;
    DWORD  fccType;
    DWORD  fccHandler;
    DWORD  dwFlags;
    DWORD  dwVersion;
    DWORD  dwVersionICM;
    char  szName[16];
    char  szDescription[128];
    char  szDriver[128];
 } ICINFO;
```

Set the **dwSize** field to the size of the ICINFO structure.

Set the **fccType** field to the four-character code to 'vidc' for video streams.

Set **fccHandler** to the four-character code identifying your driver. Your driver should use the four-character code used to install your driver and used in the description line of the .DEF file.

Specify the version number of the driver in the **dwVersion** field.

Set the **dwVersionICM** field to 1.0 (0x00010000). This specifies the version of the compression manager supported by this driver.

Use the **szName[16]** field to specify the short name of the compressor. The null-terminated name should be suitable for use in list boxes.

Use the **szDescription[128]** field to specify a null-terminated string containing a long name for the compressor.

Your driver will not normally use the **szDriver[128]** field. This field is used to specify the module that contains the driver.

Set the flags corresponding to the capabilities of your driver in the low-ordered word of the **dwFlags** field. You can use the high-ordered word for driver-specific flags. The following flags are defined for video compression and decompression drivers:

VIDCF_QUALITY

The driver supports quality levels.

VIDCF_CRUNCH

The driver supports compressing to an arbitrary frame size.

VIDCF_TEMPORAL

The driver supports inter-frame compression.

VIDCF_DRAW

The driver supports drawing to hardware with the ICM_DRAW messages.

VIDCF_FASTTEMPORAL

The driver can do temporal compression and doesn't need the previous frame.

Configuration Messages for Compression Quality

The client-application sends the following messages to obtain and set image quality values:

ICM_GETDEFAULTQUALITY

This message obtains the default quality settings of the driver.

ICM_GETQUALITY

This message obtains the current driver quality settings.

ICM_SETQUALITY

This message sets the driver quality settings.

For the video compression and decompression interface, quality is indicated by an integer ranging from 0 to 10,000. A quality level of 7,500 typically indicates an acceptable image quality. A quality level of 0 typically indicates a very low quality level (possibly even a totally black image). As the quality level moves from an acceptable level to low quality, the image might have a loss of color as the colors in the color table are merged, or as the color resolution of each pixel decreases. If your driver supports temporal compression (it

needs information from the previous frame to decompress the current frame), low and high quality might imply how much this type of compression can degrade image quality. For example, your driver might limit the compression of a high quality image to preserve sharp detail and color fidelity. Conversely, your driver might sacrifice these qualities to obtain very compressed output files.

If your driver supports quality values, it maps the values to its internal definitions used by the compression algorithms. Thus, the definition of image quality will vary from driver to driver, and, quite possibly, from compression algorithm to compression algorithm. Even though the values are not definitive, your driver should support as many individual values as possible.

The client-application obtains the capabilities for compression quality with the ICM_GETDEFAULTQUALITY and ICM_GETQUALITY messages. If your driver supports quality levels, it should respond to the ICM_GETDEFAULTQUALITY message by returning a value between 0 and 10,000 that corresponds to a good default quality level for your compressor. Your should return the current quality level for the ICM_GETQUALITY message.

The client-application sends the ICM_SETQUALITY message to set the quality level of your driver. Your driver should pass the quality value directly to the compression routine.

If your driver supports quality levels, it should set the VIDCF_QUALITY flag when it responds to the ICM_GETINFO message.

Configuration Messages for Key Frame Rate and Buffer Queue

The client-application sends the following messages to obtain the key frame rate and size of the buffer queue desired by the device driver:

ICM_GETDEFAULTKEYFRAMERATE

This message obtains the default key frame rate of the driver used during compression.

ICM_GETBUFFERSWANTED

This message obtains the number of buffers the driver wants for pre-buffering when drawing data.

The client-application uses ICM_GETDEFAULTKEYFRAMERATE to obtain the drivers recommendation for the key frame spacing for compressing data. (A key frame is a frame in a video sequence that does not require information from a previous frame for decompression.) If the client-application does not specify another value, this value determines how frequently the client-application sends an uncompressed image to your driver with the ICM_COMPRESS_KEYFRAME flag set. If your driver supports this

option, it should specify the key frame rate in the DWORD pointed to by *dwParam1* and return ICERR_OK. If it does not support this option, return ICERR_UNSUPPORTED.

The client-application uses ICM_GETBUFFERSWANTED to determine if your driver wants to maintain a queue of buffers. Your driver might maintain a queue of buffers if it renders the decompressed data and it wants to keep its hardware pipelines full. If your driver supports this option, it should specify the number of buffers in the DWORD pointed to by *dwParam1* and return ICERR_OK. If it does not support this option, return ICERR_UNSUPPORTED.

Video Compression and Decompression Messages

This section discusses the driver specific messages for video compression and decompression. The messages are covered by the three basic operations of these drivers: video compression, video decompression using the client-application, and video decompression directly to video hardware. Because video compression and decompression drivers typically use AVI files and bitmaps, this section includes a brief overview of the AVI RIFF format, the BITMAPINFO data structure, and the BITMAPINFOHEADER data structure.

About the AVI File Format

Many of the video compression and decompression messages rely on information embedded in the AVI RIFF file. Drivers do not typically access this information directly. They rely on the client-application to read and write the AVI file and maintain the RIFF file structure. While your driver should not have to manipulate an AVI file, understanding its structure helps identify the purpose of the information your driver will supply and receive.

AVI files have the following general structure:

```
RIFF('AVI'
   LIST('hdrl'
      avih(<<MainAVIHeader>>)
       LIST ('strl'
         strh(<<Stream header>>)
         strf(<<Stream format>>)
         strd(<<Stream data>>)
      )
   )

   LIST('movi'
    '00??'(<<driver Data>>)
      .
      .
      .
   '00??'(<<driver Data>>)
   )
  ' idx1'(<<AVIIndex>>)
)
```

The following table summarizes the entries in the AVI RIFF file:

RIFF Chunk	Description
RIFF 'AVI '	Identifies the file as AVI RIFF file.
LIST 'hdrl'	Identifies a chunk containing subchunks that define the format of the data.
'avih'	Identifies a chunk containing general information about the file. This includes the number of streams and the width and height of the AVI sequence.
LIST 'strl'	Identifies a chunk containing subchunks that describe the streams in a file. This chunk exists for each stream.
'strh'	Identifies a chunk containing a stream header. This includes the type of stream.
'strf'	Identifies a chunk describing the format of the data in the stream. For video streams, the information in this

	chunk is a BITMAPINFO structure. It includes palette information if appropriate.
'strd'	Identifies a chunk containing information used by compressor and decompressors. For video compressors and decompressors, this includes the state formation.
LIST 'movi '	Identifies a chunk containing subchunks used for the audio and video data.
'00??'	Identifies a chunk containing the audio or video data. For this example, both the zeros (00) and the question marks (??) are used as place holders. The zeros are replaced by stream numbers. The question marks are replaced by codes indicating the type of data in the chunk. For example, a stream for a compressed DIB might use '01dc'.
'idx1'	Identifies a chunk containing the file index.

For more information on the AVI file format, see Chapter 4, “AVI Files.”

Identifying Compression Formats

The BITMAPINFO data structure defined by Windows is used with many of the compression and decompression messages to pass information about the bitmaps being compressed and decompressed. This structure has the following fields:

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD bmiColors[];
 } BITMAPINFO;
```

The **bmiColors** field is used for the color table. The BITMAPINFOHEADER data defined for the **bmiHeader** field is used to pass information about the format of the bitmaps being compressed and decompressed. This structure has the following fields:

```
typedef struct tagBITMAPINFOHEADER {
    DWORD biSize;
    LONG  biWidth;
    LONG  biHeight;
    WORD  biPlanes;
    WORD  biBitCount;
    DWORD biCompression;
    DWORD biSizeImage;
    LONG  biXPelsPerMeter;
    LONG  biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;
```

The **biCompression** field specifies the type of compression used or requested. Windows defines the following compression formats:

BI_RGB

Specifies the bitmap is not compressed.

BI_RLE8

Specifies a run-length encoded format for bitmaps with 8 bits per pixel.

BI_RLE4

Specifies a run-length encoded format for bitmaps with 4 bits per pixel.

Extensions to the BI_RGB format include 16 and 32 bits per pixel bitmap formats. These formats do not use a color table. They embed the colors in the WORD or DWORD representing each pixel.

The 16 bit BI_RGB format is identified by setting **biCompression** to BI_RGB and setting **biBitCount** to 16. For this format, each pixel is represented by a 16-bit RGB color value. The high-bit of this value is zero. The remaining bits are divided into 3 groups of 5-bits to represent the red, green, and blue color values.

The 32 bit BI_RGB format is identified by setting **biCompression** to BI_RGB and setting **biBitCount** to 32. For this format, each pixel is represented by a 32 bit (4 byte) RGB color value. One byte is used for each red, green, and blue color value. The fourth byte is set to zero.

Your driver should support the BI_RGB format for 8 bit per pixel bitmaps. If practical, it should also support this format for 16, 24, and 32 bits per pixel bitmaps.

In addition to the new BI_RGB formats, the BI_BITFIELDS format adds new compression capabilities. This format specifies a bitmap is not compressed and color masks are defined in the **bmiColors** field of the BITMAPINFO data structure. The first DWORD in the **bmiColors** field is the red mask, the second DWORD is the green mask, and the third DWORD is the blue mask.

Your driver can also extend the format set by defining custom formats. Custom formats

use a four character code for the format in the **biCompression** field in place of the standard constants. Your driver can use a custom format to support a unique or nonstandard compression type. When you define a custom format, you can specify values other than 1, 4, 8, 16, 24, or 32 for the **biBitCount** field.

For more information about the new formats and registering custom formats, see Chapter 5, "DIB Format Extensions for Microsoft Windows." For more information about the existing formats, see the *Microsoft Windows Programmer's Reference*.

Decompressing Video Data

The client-application sends a series of messages to your driver to coordinate decompressing video data. The coordination involves the following activities:

- Setting the driver state
- Specifying the input format and determining the decompression format
- Preparing to decompress video
- Decompressing the video
- Ending decompression

The following messages are used by video compression and decompression drivers for these decompression activities:

ICM_DECOMPRESS

This message tells the driver to decompress a frame of data into a buffer provided by the client-application.

ICM_DECOMPRESS_BEGIN

This message tells the driver to prepare for decompressing data.

ICM_DECOMPRESS_END

This message tells the driver to clean up after decompressing.

ICM_DECOMPRESS_GET_FORMAT

This message asks the driver to suggest a good format for the decompressed data.

ICM_DECOMPRESS_QUERY

This message asks the driver if it can decompress a specific input format.

ICM_DECOMPRESS_GET_PALETTE

This message asks the driver to return the color table of the output data structure.

The video decompressed with these messages is returned to the client-application and it handles the display of data. If you want your driver to control the video timing or directly update the display, use the ICM_DRAW messages explained in “Decompressing Directly to Video Hardware.” If you return the decompressed video to the client-application, your driver can decompress data using either software or hardware with the ICM_DECOMPRESS messages.

Setting the Driver State

The client-application restores the driver state by sending ICM_SETSTATE. The client-application recalls the state information from the 'strd' data chunk of the AVI file. (The information was originally obtained with the ICM_GETSTATE message.) The client-application does not validate any data in the state information. It simply transfers the state information it reads from the 'strd' data chunk to your driver.

The client-application sends the information to your driver in a buffer pointed to by *dwParam1*. The size of the buffer is specified in *dwParam2*. The organization of the data in the buffer is driver dependent. If *dwParam1* is NULL, your driver should return to its default state.

Note: All information required for decompressing the image data should be part of the format data. Only optional compression parameters can be included with the state information.

Specifying the Input Format and Determining the Decompression Format

Depending on how the client-application will use the decompressed data, it will send either ICM_DECOMPRESS_GET_FORMAT or ICM_DECOMPRESS_QUERY to specify the input format and determine the decompression format. The client-application sends ICM_DECOMPRESS_GET_FORMAT if it wants your driver to suggest the decompressed format. The client-application sends ICM_DECOMPRESS_QUERY to determine if your driver supports a format it is suggesting.

Both messages send a pointer to a BITMAPINFO data structure in *dwParam1*. This structure specifies the format of the incoming compressed data. The input format was obtained by the client-application from the 'strf' chunk in the AVI file. While the output format is specified by *dwParam2*, your driver must use the message to determine how the parameter is defined.

If your driver gets ICM_DECOMPRESS_GET_FORMAT, both *dwParam1* and *dwParam2* point to BITMAPINFO data structures. The input format data is contained in the *dwParam1* structure. Your driver should fill in the *dwParam2* BITMAPINFO with information about the format it will use to decompress the data. If your driver can handle the format, return the number of bytes used for the *dwParam2* structure as the return value. If your driver cannot handle the input format, or the input format from the 'strf' chunk is incorrect, your driver should return ICERR_BADFORMAT to fail the message.

If you have format information in addition to that specified in the BITMAPINFOHEADER structure, you can add it immediately after this structure. If you do this, update the **biSize** field to specify the number of bytes used by the structure and your additional information. If a color table is part of the BITMAPINFO information, it follows immediately after your additional information. Return ICERR_OK when your driver has finished updating the data format.

If your driver gets ICM_DECOMPRESS_QUERY, *dwParam1* points to a BITMAPINFO data structure containing the input format data. The *dwParam2* parameter will either be NULL or contain a pointer to a BITMAPINFO structure describing the decompressed format the client-application wants to use.

If *dwParam2* is NULL, your decompression driver can use any output format. In this case, the client-application wants to know if you can decompress the input format and it doesn't care about the output format. If *dwParam2* points to a BITMAPINFO structure, the suggested format will be the native or best format for the decompressed data. For example, if playback is on an 8-bit device, the client-application will suggest an 8-bit DIB.

If your driver supports the specified input and output format (which might also include stretching the image), or it supports the specified input with NULL specified for *dwParam2*, return ICERR_OK to indicate the driver accepts the formats.

Your driver does not have to accept the formats suggested. If you fail the message by returning ICERR_BADFORMAT, the client-application will suggest alternate formats until your driver accepts one. If your driver exhausts the list of formats normally used, the client-application requests a format with ICM_DECOMPRESS_GET_FORMAT.

If you are decompressing to 8-bit data, your driver will also receive the ICM_DECOMPRESS_GET_PALETTE message. Your driver should add a color table to the BITMAPINFO data structure and specify the number of palette entries in the **biClrUsed** field. The space reserved for the color table will always be 256 colors.

Preparing to Decompress Video

When the client-application is ready, it sends the ICM_DECOMPRESS_BEGIN message to the driver. The client-application sets *dwParam1* and *dwParam2* to the BITMAPINFO data structures describing the input and output formats. If either of the formats is incorrect, your driver should return ICERR_BADFORMAT. Your driver should create any tables and allocate any memory that it needs to decompress data efficiently. When done, return ICERR_OK.

Decompressing the Video

The client-application sends ICM_DECOMPRESS each time it has an image to decompress. The client-application uses the flags in the file index to ensure the initial frame in a decompression sequence is a key frame.

The ICDECOMPRESS data structure specified in *dwParam1* contains the decompression

parameters. The value specified in *dwParam2* specifies the size of the structure. The ICDECOMPRESS data structure has the following fields:

```
typedef struct {
    DWORD   dwFlags;
    LPBITMAPINFOHEADER  lpbiInput;
    LPVOID  lpInput;
    LPBITMAPINFOHEADER  lpbiOutput;
    LPVOID  lpOutput;
    DWORD   ckid
 } ICDECOMPRESS;
```

The format of the input data is specified in a BITMAPINFOHEADER structure pointed to by **lpbiInput**. The input data is in a buffer specified by **lpInput**. The **lpbiOutput** and **lpOutput** fields contain pointers to the format data and buffer used for the output data.

The client-application sets the ICDECOMPRESS_HURRYUP flag in the **dwFlags** field if it wants your driver to try and decompress the data at a faster rate. The client-application will not display any data decompressed with this flag. This might let your driver avoid decompressing a frame or data, or let it minimally decompress when it needs information from this frame to prepare for decompressing a following frame.

Ending Decompression

Your driver receives ICM_DECOMPRESS_END when the client-application no longer needs data decompressed. For this message, your driver should free the resources it allocated for the ICM_DECOMPRESS_BEGIN message.

Other Messages Received During Decompression

Decompression drivers also receive the ICM_DRAW_START and ICM_DRAW_STOP messages. These messages tell the driver when the client-application starts and stops drawing the images. Most decompression drivers can ignore these messages.

Compressing Video Data

Similar to decompressing video data, your driver will receive a series of messages when it is used to compress data. The client-application will send messages to your driver to coordinate the following activities:

- Obtaining the driver state
- Specifying the input format and determining the compression format
- Preparing to compress video
- Compressing the video
- Ending compression

The following messages are used by video compression drivers:

ICM_COMPRESS

This message tells the driver to compress a frame of data into the buffer provided by the client-application.

ICM_COMPRESS_BEGIN

This message tells the driver to prepare for compressing data.

ICM_COMPRESS_END

This message tells the driver to clean up after compressing.

ICM_COMPRESS_GET_FORMAT

This message asks the driver to suggest the output format of the compressed data.

ICM_COMPRESS_GET_SIZE

This message requests the maximum size of one frame of data when it is compressed in the output format.

ICM_COMPRESS_QUERY

This message asks the driver if it can compress a specific input format.

The video compressed with these messages is returned to the client-application. When compressing data, your driver can use either software or hardware to do the compression.

Note: When AVI recompresses a file, each frame is decompressed to a full frame before it is passed to the compressor.

Obtaining the Driver State

The client-application obtains the driver state by sending ICM_GETSTATE. The client-application determines the size of the buffer needed for the state information by sending this message with *dwParam1* set to NULL. Your driver should respond to the message by returning the size of the buffer it needs for state information.

After it determines the buffer size, the client-application resends the message with *dwParam1* pointing to a block of memory it allocated. The *dwParam2* parameter specifies the size of the memory block. Your driver should respond by filling the memory with its state information. If your driver uses state information, include only optional decompression parameters with the state information. State information typically includes the setup specified by user with the ICM_CONFIGURE dialog box. Any information required for decompressing the image data must be included with the format data. When done, your driver should return the size of the state information.

The client-application does not validate any data in the state information. It simply stores

the state information in the 'strd' data chunk of the AVI file.

Specifying the Input Format and Determining the Compression Format

The client-application uses the ICM_COMPRESS_GET_FORMAT or ICM_COMPRESS_QUERY message to specify the input format and determine the compression (output) format. The client-application sends ICM_COMPRESS_GET_FORMAT if it wants your driver to suggest the compressed format. The client-application sends ICM_COMPRESS_QUERY to determine if your driver supports a format it is suggesting.

Both messages have a pointer to a BITMAPINFO data structure in *dwParam1*. This structure specifies the format of the incoming uncompressed data. The contents of *dwParam2* depends on the message.

If the client-application wants your driver to suggest the format, it determines the size of the buffer needed for the compressed data format by sending ICM_COMPRESS_GET_FORMAT. When requesting the buffer size, the client-application uses *dwParam1* to point to a BITMAPINFO structure and sets *dwParam2* to NULL. Based on the input format, your driver should return the number of bytes needed for the format buffer. Return a buffer size at least large enough to hold a BITMAPINFOHEADER data structure and a color table.

The client-application gets the output format by sending ICM_COMPRESS_GET_FORMAT with valid pointers to BITMAPINFO structures in both *dwParam1* and *dwParam2*. Your driver should return the output format in the buffer pointed to by *dwParam2*. If your driver can produce multiple formats, the format selected by your driver should be the one that preserves the greatest amount of information rather than one that compresses to the most compact size. This will preserve image quality if the video data is later edited and recompressed.

The output format data becomes the 'strf' chunk in the AVI RIFF file. The data must start out like a BITMAPINFOHEADER data structure. You can include any additional information required to decompress the file after the BITMAPINFOHEADER data structure. A color table (if used) follows this information.

If you have format data following the BITMAPINFOHEADER structure, update the **biSize** field to specify the number of bytes used by the structure and your additional data. If a color table is part of the BITMAPINFO information, it follows immediately after your additional information.

If your driver cannot handle the input format, it returns ICMERR_BADFORMAT to fail the message.

If your driver gets ICM_COMPRESS_QUERY, the *dwParam1* parameter points to a BITMAPINFO data structure containing the input format data. The *dwParam2* parameter will either be NULL or contain a pointer to a BITMAPINFO structure describing the compressed format the client-application wants to use. If *dwParam2* is NULL, your

compression driver can use any output format. (The client-application just wants to know if your driver can handle the input.) If *dwParam2* points to a BITMAPINFO structure, the client-application is suggesting the output format.

If your driver supports the specified input and output format, or it supports the specified input with NULL specified for *dwParam2*, return ICERR_OK to indicate the driver accepts the formats. Your driver does not have to accept the suggested format. If you fail the message by returning ICERR_BADFORMAT, the client-application suggests alternate formats until your driver accepts one. If your driver exhausts the list of formats normally used, the client-application requests a format with ICM_COMPRESS_GET_FORMAT.

Initialization for the Compression Sequence

When the client-application is ready to start compressing data, it sends the ICM_COMPRESS_BEGIN message. The client-application uses *dwParam1* to point to the format of the data being compressed, and uses *dwParam2* to point to the format for the compressed data. If your driver cannot handle the formats, or if they are incorrect, your driver should return ICERR_BADFORMAT to fail the message.

Before the client-application starts compressing data, it sends ICM_COMPRESS_GET_SIZE. For this message the client-application uses *dwParam1* to point to the input format and uses *dwParam2* to point to the output format. Your driver should return the worst case size (in bytes) that it expects a compressed frame to occupy. The client-application uses this size value when it allocates buffers for the compressed video frame.

Compressing the Video

The client-application sends ICM_COMPRESS for each frame it wants compressed. It uses *dwParam1* to point to an ICCOMPRESS structure containing the parameters used for compression. Your driver uses the buffers pointed to by the fields of ICCOMPRESS for returning information about the compressed data.

Your driver returns the actual size of the compressed data in the **biSizeImage** field in the BITMAPINFOHEADER data structure pointed to by the **lpbiOutput** field of ICCOMPRESS. The ICCOMPRESS data structure has the following fields:

```
typedef struct {
    DWORD            dwFlags;
    LPBITMAPINFOHEADER  lpbiOutput;
    LPVOID           lpOutput;
    LPBITMAPINFOHEADER  lpbiInput;
    LPVOID           lpInput;
    LPDWORD            lpckid;
    LPDWORD            lpdwFlags;
    LONG             lFrameNum;
    DWORD            dwFrameSize;
    DWORD            dwQuality;
    LPBITMAPINFOHEADER  lpbiPrev;
    LPVOID           lpPrev;
 } ICCOMPRESS;
```

The format of the input data is specified in a BITMAPINFOHEADER structure pointed to by **lpbiInput**. The input data is in a buffer specified by **lpInput**. The **lpbiOutput** and **lpOutput** fields contain pointers to the format data and buffer used for the output data. Your driver must indicate the size of the compressed video data in the **biSizeImage** field in the BITMAPINFO structure specified for **lpbiOutput**.

The **dwFlags** field specifies flags used for compression. The client-application sets ICM_COMPRESS_KEYFRAME flag if the input data should be treated as a key frame. (A key frame is one that does not require data from a previous frame for decompression.) When this flag is set, your driver should treat the image as the initial image in a sequence.

The **lpckid** field specifies a pointer to a buffer used to return the chunk ID for data in the AVI file. Your driver should assign a two-character code for the chunk ID only if it uses a custom chunk ID. For more information on chunk IDs, see Chapter 4, “AVI Files.”

The **lpdwFlags** field specifies a pointer to a buffer used to return flags for the AVI index. The client-application will add the returned flags to the file index for this chunk. If the compressed frame is a key frame (a frame that does not require a previous frame for decompression), your driver should set the AVIIF_KEYFRAME flag in this field. Your driver can define its own flags but they must be set in the high word only.

The **lFrameNum** field specifies the frame number of the frame to compress. If your driver is performaing fast temporal compression, check this field to see if frames are being sent out of order or if the client-application is having a frame recompressed.

The **dwFrameSize** field indicates the maximum size (in bytes) desired for the compressed frame. If it specifies zero, your driver determines the size of the compressed image. If it is non-zero, your driver should try to compress the frame to within the specified size. This might require your driver to sacrifice image quality (or make some other trade-off) to obtain the size goal. Your driver should support this if it sets the VIDCF_CRUNCH flag when it responds to the ICM_GETINFO message.

The **dwQuality** field specifies the compression quality. Your driver should support this if it sets the VIDCF_QUALITY flag when it responds to the ICM_GETINFO message.

The format of the previous data is specified in a BITMAPINFOHEADER structure pointed to by **lpbiPrev**. The input data is in a buffer specified by **lpPrev**. Your driver will use this information if it performs temporal compression (that is, it needs the previous

frame to compress the current frame). If your driver supports temporal compression, it should set the VIDCF_TEMPORAL flag when it responds to the ICM_GETINFO message. If your driver supports temporal compression and does not need the information in the **lpbiPrev** and **lpPrev** fields, it should set the VIDCF_FASTTEMPORAL flag when it responds to the ICM_GETINFO message. The VIDCF_FASTEMPORAL flag can decrease the processing time because your driver does not need to access data specified in **lpbiPrev** and **lpPrev**.

When your driver has finished decompressing the data, it returns ICERR_OK.

Ending Compression

Your driver receives ICM_COMPRESS_END when the client-application no longer needs data compressed, or when the client-application is changing the format or palette. After sending ICM_COMPRESS_END, the client-application must send ICM_COMPRESS_BEGIN to continue compressing data. Your driver should not expect the client-application to send a ICM_COMPRESS_END message for each ICM_COMPRESS_BEGIN message. The client-application can use ICM_COMPRESS_BEGIN to restart compression without sending ICM_COMPRESS_END.

When the driver is no longer needed, the system will close it by sending DRV_CLOSE.

Decompressing Directly to Video Hardware

Drivers that can render video directly to hardware should support the ICM_DRAW messages in addition to the ICM_DECOMPRESS messages. The ICM_DRAW messages decompress data directly to hardware rather than into a data buffer returned to the client-application by the decompression driver.

Your driver will receive a series of messages from the client-application to coordinate the following activities to decompress a video sequence:

- Setting the driver state
- Specifying the input format
- Preparing to decompress video
- Decompressing the video
- Ending decompression

The following ICM_DRAW messages are used by video decompression drivers for these decompression activities:

ICM_DRAW

This message tells the driver to decompress a frame of data and draw it to the screen.

ICM_DRAW_BEGIN

This message tells the driver to get ready to draw data.

ICM_DRAW_END

This message tells the driver to clean up after decompressing an image to the screen.

ICM_DRAW_REALIZE

This message realizes a palette.

ICM_DRAW_QUERY

This message determines if the driver can render data in a specific format.

The video decompressed with the ICM_DRAW messages is retained by your driver and it handles the display of data. These messages control only the decompression process. The messages used to control the drawing are described separately. Your driver will receive the ICM_DRAW messages only if it sets the VIDCF_DRAW flag when it responds to the ICM_GETINFO message.

Setting the Driver State

The client-application restores the driver state by sending ICM_SETSTATE. The process for handling this message is the same as for the ICM_DECOMPRESS messages.

Specifying the Input Format

Because your driver handles the drawing of video, the client-application does not need to determine the output format. The client-application needs to know only if your driver can handle the input format. It sends ICM_DRAW_QUERY to determine if your driver supports the input format. The input format is specified with a pointer to a BITMAPINFO data structure in *dwParam1*. The *dwParam2* parameter is not used.

If your driver supports the specified input format, return ICERR_OK to indicate the driver accepts the formats. If your driver does not support the format, return ICERR_BADFORMAT.

Preparing to Decompress Video

When the client-application is ready, it sends the ICM_DRAW_BEGIN message to the driver to prepare the driver for decompressing the video stream. Your driver should create any tables and allocate any memory that it needs to decompress data efficiently.

The client-application sets *dwParam1* to the ICDRAWBEGIN data structure. The size of this structure is contained in *dwParam2*. The ICDRAWBEGIN structure has the following fields:

```
typedef struct {
    DWORD            dwFlags;
    HPALETTE         hpal;
    HWND             hwnd;
    HDC              hdc;
    int              xDst;
    int              yDst;
    int              dxDst;
    int              dyDst;
    LPBITMAPINFOHEADER  lpbi;
    int              xSrc;
    int              ySrc;
    int              dxSrc;
    int              dySrc;
    DWORD            dwRate;
    DWORD            dwScale;
 } ICDRAWBEGIN;
```

The **dwFlags** field can specify any of the following flags:

ICDRAW_QUERY

Set when the client-application wants to determine if the driver can handle the decompression. The driver does not actually decompress the data.

ICDRAW_FULLSCREEN

Indicates the full screen is used to draw the decompressed data.

ICDRAW_HDC

Indicates a window and DC are used to draw the decompressed data.

If the ICDRAW_QUERY flag is set, the client-application is interrogating your driver to determine if can decompress the data with the parameters specified in the ICDRAWBEGIN data structure. Your driver should return ICM_ERR_OK if it can accept the parameters. It should return ICM_ERR_NOTSUPPORTED if it does not accept them.

When the ICDRAW_QUERY flag is set, ICM_DRAW_BEGIN will not be paired with ICM_DRAW_END. Your driver will receive another ICM_DRAW_BEGIN without this flag to start the actual decompression sequence.

The ICDRAW_FULLSCREEN and ICDRAW_HDC flags are described with the data structure fields associated with them.

Your driver can ignore the palette handle specified in the **hpal** field.

The **hwnd** and **hdc** field specifies the handle of the window and DC used for drawing. These fields are valid only if the ICDRAW_HDC flag is set in the **dwFlags** field.

The **xDst** and **yDst** fields specify the x- and y-position of the upper-right corner of the destination rectangle. (This is relative to the current window or display context.) The **dxDst** and **dyDst** fields specifies the width and height of the destination rectangle. These

fields are valid only if ICDRAW_HDC flag is set. The ICDRAW_FULLSCREEN flag indicates the entire screen should be used for display and overrides any values specified for these fields.

The **xSrc**, **ySrc**, **dxSrc**, and **dySrc** fields specify a source rectangle used to clip the frames of the video sequence. The source rectangle is stretched to fill the destination rectangle. The **xSrc** and **ySrc** fields specify x- and y-position of the upper-right corner of the source rectangle. (This is relative to a full frame image of the video.) The **dxSrc** and **dySrc** fields specify the width and height of the source rectangle.

Your driver should stretch the image from the source rectangle to fit the destination rectangle. If the client-application changes the size of the source and destination rectangles, it will send the ICM_DRAW_END message and specify new rectangles with a new ICM_DRAW_BEGIN message. For more information on handling the source and destination rectangles, see the **StretchDIBits** function in the *Microsoft Windows Programmer's Reference*.

The **lpbi** field specifies a pointer to a **BITMAPINFOHEADER** data structure containing the input format.

The **dwRate** field specifies the decompression rate in an integer format. To obtain the rate in frames-per-second divide this value by the value in *dwScale*. Your driver will use these values when it handles the ICM_DRAW_START message.

If your driver can decompress the data with the parameters specified in the ICDRAWBEGIN data structure, your driver should return ICERR_OK and allocate any resources it needs to efficiently decompress the data. If your driver cannot decompress the data with the parameters specified, your driver should fail the message by returning ICERR_NOTSUPPORTED. When this message fails, your driver will not get an ICM_DRAW_END message so it should not prepare its resources for other ICM_DRAW messages.

Decompressing the Video

The client-application sends ICM_DRAW each time it has an image to decompress. (Your driver should use this message to decompress images. It should wait for the ICM_DRAW_START message before it begins to render them.) The client-application uses the flags in the file index to ensure the first frame in a series of frames decompressed starts with a key frame boundary. Your driver must allocate the memory it needs for the decompressed image.

The ICDRAW data structure specified in *dwParam1* contains the decompression parameters. The value specified in *dwParam2* specifies the size of the structure. The ICDRAW data structure has the following fields:

```
typedef struct {
    DWORD   dwFlags;
    LPVOID  lpFormat;
    LPVOID  lpData;
    DWORD   cbData;
} ICDRAW;
```

The format of the input data is specified in a BITMAPINFOHEADER structure pointed to by **lpFormat**. The input data is in a buffer specified by **lpData**. The number of bytes in the input buffer is specified by **cbData**.

The client-application sets the ICDRAW_HURRYUP flag in the **dwFlags** field when it wants your driver to try to decompress data at a faster rate. For example, the client-application might use this flag when the video is starting to lag behind the audio. If your driver cannot speed up its decompression and rendering performance, it might be necessary to avoid rendering a frame of data. The client-application sets the ICDRAW_UPDATE flag and sets **lpData** to NULL if it wants your driver to update the screen based on data previously received.

When your driver has finished decompressing the data, it returns ICERR_OK. After the driver returns from this message, the client-application deallocates or reuses the memory containing the format and image data. If your driver needs the format or image data for future use, it should copy the data it needs before it returns from the message.

Ending Decompression

Your driver receives ICM_DRAW_END when the client-application no longer needs data decompressed or rendered. For this message, your driver should free the resources it allocated for the ICM_DRAW_BEGIN message. Your driver should also leave the display in the full screen mode.

After sending ICM_DRAW_END, the client-application must send ICM_DRAW_BEGIN to continue decompressing data. Your driver should not expect the client-application to send a ICM_DRAW_END message for each ICM_DRAW_BEGIN message. The client-application can use ICM_DRAW_BEGIN to restart decompression without sending ICM_DRAW_END.

Rendering the Data

The client-application sends the following messages to control the driver’s internal clock for rendering the decompressed data:

ICM_DRAW_GETTIME

This message obtains the value of the driver’s internal clock if it is handling the timing of drawing frames.

ICM_DRAW_SETTIME

This message sets the driver's internal clock if it is handling the timing of drawing frames.

ICM_DRAW_START

This message tells the driver to start its internal clock if it is handling the timing of drawing frames.

ICM_DRAW_STOP

This message tells the driver to stop its internal clock if it is handling the timing for drawing frames.

ICM_DRAW_WINDOW

This message tells the driver that the display window has been moved, hidden, or displayed.

ICM_DRAW_FLUSH

This message tells the driver to flush any frames that are waiting to be drawn.

The client-application sends ICM_DRAW_START to have your driver start (or continue) rendering data at the rate specified by the ICM_DRAW_BEGIN message. The ICM_DRAW_STOP message pauses the internal clock. Neither of these messages use *dwParam1*, *dwParam2*, or a return value.

The client-application uses ICM_DRAW_GETTIME to obtain the value of the internal clock. Your driver returns the current time value (this is normally frame numbers for video) in the DWORD indicated by the pointer specified by *dwParam1*. The current time is relative to the start of drawing.

The client-application uses ICM_DRAW_SETTIME to set the value of the internal clock. Typically, the client-application uses this message to synchronize the driver's clock to an external clock. Your driver should set its clock to the value (this is normally frame numbers for video) specified in the DWORD pointed to by *dwParam1*.

The client-application sends ICM_DRAW_FLUSH to have your driver discard any frames that have not been drawn.

Using Installable Compressors for Non-video Data

Installable compressors are not necessarily limited to video data. By using a different value than 'vidc' in the **fccType** field, you can specify that your installable driver expects

to handle a type of data that is not video. (Four-character codes for non-video data should also be registered. See the "Architecture of a Video Compression and Decompression Driver" section for information on registering four-character codes.)

While VidEdit does not support data that is not audio or video, MCIAVI does provide limited support for other data types using installable renderers. If you create a stream with a four-character code type that does not represent audio or video, its type and handler information will be used to search for a driver capable of handling the data. The search will follow the same procedure used for installable compressor drivers.

Writing a driver to render non-video data is very similar to rendering video, with the following differences:

- The format used is not a BITMAPINFO structure. The format is defined by the class of decompressor.
- The ICM_DECOMPRESS messages are not used. All data is rendered using the ICM_DRAW messages because there is no defined decompressed form for arbitrary data.

Testing Video Compression and Decompression Drivers

You can exercise both the compression and decompression capabilities of a driver with the VidEdit editing tool. You can also exercise the decompression capabilities of a driver with MCIAVI. (One way to test the decompression capabilities is to preview an unedited file in VidEdit. For this function, VidEdit uses MCIAVI to decompress the file.)

Video Compression and Decompression Driver Reference

This section is an alphabetic reference to the messages and data structures provided by Windows for use by video compression and decompress drivers. There are separate sections for messages and data structures. The COMPDDK.H file defines the messages and data structures.

Video Compression and Decompression Driver Message Reference

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

The following messages are used by video compression and decompression drivers for compressing data:

ICM_COMPRESS

This message tells the driver to compress a frame of data into the buffer provided by the calling application.

ICM_COMPRESS_BEGIN

This message prepares the driver for compressing data.

ICM_COMPRESS_END

This message tells the driver to clean up after compressing.

ICM_COMPRESS_GET_FORMAT

This message returns the output format of the compressed data.

ICM_COMPRESS_GET_SIZE

This message obtains the maximum size of one frame of data when it is compressed in the output format.

ICM_COMPRESS_QUERY

This message asks the driver if it can compress a specific input format.

The following messages are used by video compression and decompression drivers for decompression:

ICM_DECOMPRESS

This message tells the driver to decompress a frame of data into a buffer provided by the calling application.

ICM_DECOMPRESS_BEGIN

This message prepares the driver for decompressing data.

ICM_DECOMPRESS_END

This message tells the driver to clean up after decompressing.

ICM_DECOMPRESS_GET_FORMAT

This message asks the driver to suggest the format of the decompressed data.

ICM_DECOMPRESS_GET_PALETTE

This message asks the driver to return the color table of the output data structure.

ICM_DECOMPRESS_QUERY

This message asks the driver if it can decompress a specific input format.

The following messages are used by video compression and decompression drivers for drawing with the compressed data:

ICM_DRAW

This message tells the driver to decompress a frame of data and draw it to the screen.

ICM_DRAW_BEGIN

This message tells the driver to get ready to draw data.

ICM_DRAW_END

This message tells the driver to clean up after decompressing an image to the screen.

ICM_DRAW_GETTIME

This message obtains the value of the driver's internal clock if it is handling the timing of drawing frames.

ICM_DRAW_QUERY

This message determines if the driver can render data in a specific format.

ICM_DRAW_REALIZE

This message obtains a palette from the driver.

ICM_DRAW_SETTIME

This message informs a video compression driver of what frame it should be drawing.

ICM_DRAW_START

This message tells the driver to start its internal clock if it is handling the timing of drawing frames.

ICM_DRAW_STOP

This message tells the driver to stop its internal clock if it is handling the timing for drawing frames.

ICM_DRAW_WINDOW

This message tells the driver when a window has physically moved, or has become totally obscured.

ICM_DRAW_FLUSH

This message is sent to a video compression driver to flush any frames it has that are waiting to be drawn.

The following messages are used to configure video compression and decompression drivers:

ICM_ABOUT

This message displays an about dialog box for a compressor driver.

ICM_CONFIGURE

This message displays a configuration dialog box for a compressor driver.

ICM_GETBUFFERSWANTED

This message obtains information about how much pre-buffering the driver wants.

ICM_GETDEFAULTKEYFRAMERATE

This message obtains the preferred key frame spacing of the driver.

ICM_GETDEFAULTQUALITY

This message obtains the default quality setting of the driver.

ICM_GETINFO

This message returns information about the driver.

ICM_GETQUALITY

This message obtains the current quality setting of the driver.

ICM_GETSTATE

This message fills in a compressor-specific block of memory describing the compressor’s current configuration.

ICM_SETQUALITY

This message sets the quality level of the compressor.

ICM_SETSTATE

This message sets the quality level for compression.

The following system message is used to open video compression and decompression drivers:

DRV_OPEN

This system message is sent to a video compression driver each time it is opened.

Video Compression and Decompression Driver Messages

This section contains an alphabetical list of the video compression and decompression messages that can be received and sent by video capture 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.

DRV_OPEN

This system message is sent to a video compression driver each time it is opened.

Parameters

DWORD *dwDriverIdentifier*

Specifies the handle returned to the application opening the driver.

HANDLE *hDriver*

Specifies the handle created by the system. This handle is returned to the application. A unique handle is created each time the driver is opened.

LONG *lParam1*

Specifies a pointer to a NULL-terminated string. The string contains any characters that follow the filename in the SYSTEM.INI file. If the device driver was opened by filename, or if there is no additional information, a NULL string or a NULL pointer is passed. Device drivers should verify that *lParam1* is not NULL before dereferencing it.

LONG *lParam2*

Specifies a far pointer to an **ICOPEN** structure, or zero if the driver is opened only for configuration by the Drivers option of the Control Panel. If an **ICOPEN** structure is passed, the driver should verify that the **fccType** field contains 'vidc'. This indicates the driver is opened as a video compressor.

Return Value The driver should return zero to fail the call. A non-zero return value is passed back to the driver in the ID field each time **DriverProc** is sent a message with **SendDriverMessage** or **CloseDriver**.

ICM_ABOUT

This message is sent to a video compression driver to display its about dialog box.

Parameters DWORD *dwParam1*

Specifies an **HWND** which should be the parent of the displayed dialog box.

If *dwParam1* is -1, the driver should return ICERR_OK if it has an about dialog box, however, the driver should not display it. The driver should return ICERR_UNSUPPORTED if it does not display a dialog box.

DWORD *dwParam2*

Not Used.

Return Value Return ICERR_OK if the driver supports this message. Otherwise, return ICERR_UNSUPPORTED.

See Also ICM_CONFIGURE, ICM_GETINFO

ICM_COMPRESS

This message is sent to a video compression driver to compress a frame of data into the application-supplied buffer.

Parameters DWORD *dwParam1*

Specifies a far pointer to an **ICCOMPRESS** data structure. The following fields of the **ICCOMPRESS** specify the compression parameters:

The **lpbiInput** field of **ICCOMPRESS** contains the format of the uncompressed data; the data itself is in a buffer pointed to by **lpInput**.

The **lpbiOutput** field of the **ICCOMPRESS** data structure contains a pointer to the output (compressed) format, and **lpOutput** contains a pointer to a buffer used for the compressed data.

The **lpbiPrev** field of the **ICCOMPRESS** data structure contains a pointer to the format of the previous frame, and **lpPrev** contains a pointer to a buffer used for the previous data. These fields are used by drivers that do temporal compression.

The driver should use the **biSizeImage** field of the **BITMAPINFOHEADER** structure associated with **lpbiOutput** to return the size of the compressed frame.

The **lpdwFlags** field points to a DWORD. The driver should fill the DWORD with the flags that should go in the AVI index. In particular, if the returned frame is a key frame, your driver should set the AVIIF_KEYFRAME flag.

The **dwFrameSize** field contains the size the compressor should try to make the frame fit within. This size is used for compression methods that can make tradeoffs between compressed image size and image quality.

The **dwQuality** field contains the specific quality the compressor should use if it supports it.

DWORD *dwParam2*

Specifies the size of the **ICCOMPRESS** structure.

Return Value Return ICERR_OK if successful. Otherwise, return an error number.

See Also ICM_COMPRESS_BEGIN, ICM_COMPRESS_END, ICM_DECOMPRESS, ICM_DRAW

ICM_COMPRESS_BEGIN

This message is sent to a video compression driver to prepare it for compressing data.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure indicating the input format.

DWORD *dwParam2*

Specifies a far pointer to a **BITMAPINFO** data structure indicating the output format.

Return Value Return ICERR_OK if the specified compression is supported. Otherwise, return ICERR_BADFORMAT if the input or output format is not supported.

Comments The driver should set up any tables or memory that it needs to compress the data formats efficiently when it receives the **ICM_COMPRESS** messages.

ICM_COMPRESS_BEGIN and ICM_COMPRESS_END do not nest. If your driver receives an ICM_COMPRESS_BEGIN message before compression is stopped with ICM_COMPRESS_END, it should restart compression with new parameters.

See Also ICM_COMPRESS, ICM_COMPRESS_END, ICM_DECOMPRESS_BEGIN, ICM_DRAW_BEGIN

ICM_COMPRESS_END

This message is sent to a video compression driver to end compression. The driver should clean-up after compressing, and release any memory allocated during processing of an **ICM_COMPRESS_BEGIN** message.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if successful. Otherwise, return an error number.

Comments ICM_COMPRESS_BEGIN and ICM_COMPRESS_END do not nest. If your driver receives an ICM_COMPRESS_BEGIN message before compression is stopped with ICM_COMPRESS_END, it should restart compression with new parameters.

See Also ICM_COMPRESS_BEGIN, ICM_COMPRESS, ICM_DECOMPRESS_END, ICM_DRAW_END

ICM_COMPRESS_GET_FORMAT

This message is sent to a video compression driver to suggest the output format of the compressed data.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure indicating the input format.

DWORD *dwParam2*

Specifies zero or a far pointer to a **BITMAPINFO** data structure used by the driver to return the output format.

Return Value Return the size of the output format.

Comments If *dwParam2* is zero, the driver should simply return the size of the output format.

If *dwParam2* is non-zero, the driver should fill the **BITMAPINFO** data structure with the default output format corresponding to the input format specified for *dwParam1*. If the compressor can produce several different formats, the default format should be the one which will preserve the greatest amount of information.

For example, the Microsoft Video Compressor can compress 16-bit data into either an 8-bit palettized compressed form or a 16-bit true-color compressed form. The 16-bit format more accurately represents the original data, and thus is returned by this message.

See Also ICM_COMPRESS_QUERY, ICM_DECOMPRESS_GET_FORMAT, ICM_DRAW_GET_FORMAT

ICM_COMPRESS_GET_SIZE

This message is sent to a video compression driver to obtain the maximum size of one frame of data when it is compressed in the output format.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure indicating the input format.

DWORD *dwParam2*

Specifies a far pointer to a **BITMAPINFO** data structure indicating the output format.

Return Value Return the maximum number of bytes a single compressed frame can occupy.

Comments Typically, applications use this message to determine how large a buffer to allocate for the compressed frame.

The driver should calculate the size of the largest possible frame based on the input and target formats.

See Also ICM_COMPRESS_QUERY, ICM_COMPRESS_GET_FORMAT

ICM_COMPRESS_QUERY

This message is sent to a video compression driver to determine if it can compress a specific input format, or if it can compress the input format to a specific output format.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure describing the input format.

DWORD *dwParam2*

Specifies a far pointer to a **BITMAPINFO** data structure describing the output format, or zero. Zero indicates any output format is acceptable.

Return Value Return ICERR_OK if the specified compression is supported. Otherwise, return an error. The following errors are defined:

ICERR_OK

No error.

ICERR_BADFORMAT

The input or output format is not supported.

Comments On receiving this message, the driver should examine the **BITMAPINFO** structure associated with *dwParam1* to see if it can compress the input format. The driver should return ICERR_OK only if it can compress the input format to the output format specified for *dwParam2*. (If any output format is acceptable, *dwParam2* is zero.)

See Also ICM_COMPRESS_GET_FORMAT

ICM_CONFIGURE

This message is sent to a video compression driver to display its configuration dialog box.

Parameters DWORD *dwParam1*

Specifies an **HWND** which should be the parent of the displayed dialog.

If *dwParam1* is -1, the driver should return ICERR_OK if it has a configuration dialog box, however, the driver should not display it. The driver should return ICERR_UNSUPPORTED if it does not display a dialog box.

DWORD *dwParam2*

Not Used.

Return Value Return ICERR_OK if the driver supports this message. Otherwise, return ICERR_UNSUPPORTED.

Comments This message is distinct from the **DRV_CONFIGURE** message which is used for hardware configuration. This message should let the user configure the internal state referenced by **ICM_GETSTATE** and **ICM_SETSTATE**. For example, this dialog box can let the user change parameters affecting the quality level and other similar compression options.

See Also DRV_CONFIGURE, ICM_ABOUT, ICM_GETINFO

ICM_DECOMPRESS

This message is sent to a video compression driver to decompress a frame of data into an application-supplied buffer.

Parameters

DWORD *dwParam1*

Specifies a far pointer to an **ICDECOMPRESS** structure.

DWORD *dwParam2*

Specifies the size of the **ICDECOMPRESS** structure.

Return Value

Return ICERR_OK if successful. Otherwise, return an error number.

Comments

If the driver is supposed to decompress data directly to the screen instead of a buffer, it will receive the **ICM_DRAW** message rather than this one.

The driver should return an error if this message is received before the **ICM_DECOMPRESS_BEGIN** message.

See Also

ICM_COMPRESS_BEGIN, ICM_DECOMPRESS_BEGIN, ICM_DECOMPRESS_END, ICM_DRAW_BEGIN

ICM_DECOMPRESS_BEGIN

This message is sent to a video compression driver for decompressing data. When the driver receives this message, it should allocate buffers and do any time-consuming operations so that it can process **ICM_DECOMPRESS** messages efficiently.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure describing the input format.

DWORD *dwParam2*

Specifies a far pointer to a **BITMAPINFO** data structure describing the output format.

Return Value

Return ICERR_OK if the specified decompression is supported. Otherwise, return an error number. The following errors are defined:

ICERR_OK

No error.

ICERR_BADFORMAT

The input or output format is not supported.

Comments

If the calling application wants the driver to decompress data directly to the screen, it sends the **ICM_DRAW_BEGIN** message.

ICM_DECOMPRESS_BEGIN and ICM_DECOMPRESS_END do not nest. If your driver receives an ICM_DECOMPRESS_BEGIN message before decompression is

stopped with ICM_DECOMPRESS_END, it should restart decompression with new parameters.

See Also ICM_COMPRESS_BEGIN, ICM_DECOMPRESS, ICM_DECOMPRESS_END, ICM_DRAW_BEGIN

ICM_DECOMPRESS_END

This message is sent to a video compression driver to have it clean-up after decompressing.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if successful. Otherwise, return an error number.

Comments The driver should free any resources allocated in response to the **ICM_DECOMPRESS_BEGIN** message.

ICM_DECOMPRESS_BEGIN and ICM_DECOMPRESS_END do not nest. If your driver receives an ICM_DECOMPRESS_BEGIN message before decompression is stopped with ICM_DECOMPRESS_END, it should restart decompression with new parameters.

See Also ICM_COMPRESS_END, ICM_DECOMPRESS_BEGIN, ICM_DECOMPRESS, ICM_DRAW_END

ICM_DECOMPRESS_GET_FORMAT

This message is sent to a video compression driver to obtain the format of the decompressed data.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure describing the input format.

DWORD *dwParam2*

Specifies zero or a far pointer to a **BITMAPINFO** data structure used by the driver to describe the output format.

Return Value Return the size of the output format.

Comments If *dwParam2* is zero, the driver should simply return the size of the output format. Applications set *dwParam2* to zero to determine the size of the buffer it needs to allocate.

If *dwParam2* is non-zero, the driver should fill the **BITMAPINFO** data structure with the default output format corresponding to the input format specified for *dwParam1*. If the compressor can produce several different formats, the default format should be the one which will preserve the greatest amount of information.

For example, if a driver can produce either 24-bit full-color images or 8-bit gray-scale images, the default should be 24-bit images. This ensures the highest possible image quality if the video data must be edited and re-compressed.

See Also ICM_COMPRESS_GET_FORMAT, ICM_DECOMPRESS_GET_PALETTE, ICM_DECOMPRESS_QUERY

ICM_DECOMPRESS_GET_PALETTE

This message is sent to a video compression driver to have it fill in the color table of the output **BITMAPINFOHEADER** structure.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure indicating the input format.

DWORD *dwParam2*

Specifies zero or a far pointer to a **BITMAPINFO** data structure used to return the color table. The space reserved for the color table will always be at least 256 colors.

Return Value Return the size of the output format or an error code.

Comments If *dwParam2* is zero, the driver should simply return the size of the output format. Applications set this value to zero when they want to determine the size of the output format.

If *dwParam2* is non-zero, the driver should set the **biClrUsed** field of the **BITMAPINFOHEADER** data structure to the number of colors in the color table. The driver fills the **bmiColors** fields of the **BITMAPINFO** data structure with the actual colors.

The driver should support this message only if it uses a palette other than the one in the input format.

See Also ICM_DECOMPRESS_GET_FORMAT

ICM_DECOMPRESS_QUERY

This message is sent to a video compression driver to determine if the driver can decompress a specific input format, or if it can decompress the input format to a specific output format.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure describing the input format.

DWORD *dwParam2*

Specifies zero or a far pointer to a **BITMAPINFO** data structure used by the driver to describe the output format. Zero indicates that any output format is acceptable.

Return Value

Return ICERR_OK if the specified decompression is supported. Otherwise, return an error number. The following errors are defined:

ICERR_OK

No error.

ICERR_BADFORMAT

The input or output format is not supported.

See Also

ICM_COMPRESS_QUERY

ICM_DRAW

This message is sent to a video compression driver to decompress a frame of data and draw it to the screen.

Parameters

DWORD *dwParam1*

Specifies a far pointer to an **ICDRAW** structure.

DWORD *dwParam2*

Specifies the size of the **ICDRAW** structure.

Return Value

Return ICERR_OK if successful. Otherwise, return an error number.

Comments

If the **ICDRAW_UPDATE** flag is set in **dwFlags** field of the **ICDRAW** data structure, the area of the screen used for drawing is invalid and needs to be updated.

If the **ICDRAW_HURRYUP** flag is set in the **dwFlags** field, the calling application wants the driver to proceed as quickly as possible, possibly not even updating the screen.

If the **ICDRAW_PREROLL** flag is set in the **dwFlags** field, this video frame is merely

preliminary information and should not be displayed if possible. For instance, if play is to start from frame 10, and frame 0 is the nearest previous keyframe, frames 0 through 9 will have the **ICDRAW_PREROLL** flag set.

If the driver is to decompress data into a buffer instead of drawing directly to the screen, the **ICM_DECOMPRESS** message is sent instead.

See Also ICM_DECOMPRESS, ICM_DRAW_BEGIN, ICM_DRAW_END, ICM_DRAW_START, ICM_DRAW_STOP

ICM_DRAW_BEGIN

This message is sent to a video compression driver to prepare it for drawing data.

Parameters

DWORD *dwParam1*

Specifies a far pointer to a **ICDRAWBEGIN** data structure describing the input format.

DWORD *dwParam2*

Specifies the size of the **ICDRAWBEGIN** data structure describing the output format.

Return Value Return ICERR_OK if the driver supports drawing the data to the screen in the manner and format specified. Otherwise, return an error number. The following errors are defined:

ICERR_OK

No error.

ICERR_BADFORMAT

The input or output format is not supported.

ICERR_NOTSUPPORTED

The message is not supported.

Comments If the driver is supposed to decompress data into a buffer instead of drawing directly to the screen, the **ICM_DECOMPRESS_BEGIN** message is sent rather than this one.

If the driver does not support drawing directly to the screen, return ICERR_NOTSUPPORTED.

ICM_DRAW_BEGIN and ICM_DRAW_END do not nest. If your driver receives an ICM_DRAW_BEGIN message before decompression is stopped with ICM_DRAW_END, it should restart decompression with new parameters.

See Also ICM_DECOMPRESS_BEGIN, ICM_DRAW, ICM_DRAW_END, ICM_DRAW_START

ICM_DRAW_END

This message is sent to video compression drivers to clean-up after decompressing an image to the screen.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if successful. Otherwise, return an error number.

Comments ICM_DRAW_BEGIN and ICM_DRAW_END do not nest. If your driver receives an ICM_DRAW_BEGIN message before decompression is stopped with ICM_DRAW_END, it should restart decompression with new parameters.

See Also ICM_DECOMPRESS_END, ICM_DRAW, ICM_DRAW_BEGIN, ICM_DRAW_STOP

ICM_DRAW_FLUSH

This message is sent to a video compression driver to flush any frames it has that are waiting to be drawn.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value None.

Comments This message is used only by hardware which does its own asynchronous decompression, timing, and drawing.

See Also ICM_DRAW, ICM_DRAW_END, ICM_DRAW_STOP, ICM_GETBUFFERSWANTED

ICM_DRAW_GETTIME

This message is sent to a video compression driver to obtain the current value of its internal clock if it is handling the timing of drawing frames.

Parameters DWORD *dwParam1*

Specifies a far pointer to a LONG to be used by the driver to return the current time. The return value should be specified in samples. This corresponds to frames for video.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if successful.

Comments This message is generally only supported by hardware which does its own asynchronous decompression, timing, and drawing. The message will also only be sent if the hardware is being used as the synchronization master.

See Also ICM_DRAW_START, ICM_DRAW_STOP, ICM_DRAW_SETTIME

ICM_DRAW_QUERY

This message is sent to a video compression driver to determine if it can render data in a specific format.

Parameters DWORD *dwParam1*

Specifies a far pointer to a **BITMAPINFO** data structure describing the input format.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if the compressor can render data in the specified format. Otherwise, return an error number. The following errors are defined:

ICERR_OK

No error.

ICERR_BADFORMAT

The format is not supported.

Comments This message asks if the compressor recognizes the format for draw operations. The ICM_DRAW_BEGIN is sent to see if the compressor can draw the data.

See Also ICM_DECOMPRESS_QUERY

ICM_DRAW_REALIZE

This message is sent to a video compression driver to realize its palette used while

drawing.

Parameters DWORD *dwParam1*

Specifies a handle to the display context used to realize palette.

DWORD *dwParam2*

Specifies TRUE if the palette is to be realized in the background. Specifies FALSE if the palette is to be realized in the foreground.

Return Value Return ICERR_OK if palette realized.

Comments Drivers need to respond to this message only if the drawing palette is different from the decompressed palette.

If this message is not supported (returns ICERR_UNSUPPORTED), the palette associated with the decompressed data is realized.

See Also ICM_DRAW_BEGIN

ICM_DRAW_RENDERBUFFER

This message is sent to a video compression driver to tell it to draw the frames that have been passed to it.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value None.

Comments This message is typically used to perform a "seek" operation when, rather than playing a sequence of video frames, the driver must be specifically instructed to display a single video frame passed to it.

This message is used only by hardware which does its own asynchronous decompression, timing, and drawing.

See Also ICM_DRAW, ICM_DRAW_END, ICM_DRAW_START

ICM_DRAW_SETTIME

This message is sent to a video compression driver to inform it of what frame it should be drawing if it is handling the timing of drawing frames.

Parameters DWORD *dwParam1*

Specifies a LONG containing the sample which the driver should now be rendering. The value will be specified in samples. This corresponds to frames for video.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if successful.

Comments This message is generally only supported by hardware which does its own asynchronous decompression, timing, and drawing. The message will only be sent if the hardware is not being used as the synchronization master.

Typically, the driver will compare the specified "correct" value with its own internal clock, and take actions to synchronize the two if the difference is great enough.

See Also ICM_DRAW_START, ICM_DRAW_STOP, ICM_DRAW_GETTIME

ICM_DRAW_START

This message is sent to a video compression driver to start its internal clock for the timing of drawing frames.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value None.

Comments This message is typically used by hardware which does its own asynchronous decompression, timing, and drawing.

When it receives this message, the driver should start rendering data at the rate specified in the **ICM_DRAW_BEGIN** message.

ICM_DRAW_START and ICM_DRAW_STOP do not nest. If your driver receives an ICM_DRAW_START message before rendering is stopped with ICM_DRAW_STOP, it should restart rendering with new parameters.

See Also ICM_DRAW, ICM_DRAW_BEGIN, ICM_DRAW_END, ICM_DRAW_STOP

ICM_DRAW_STOP

This message is sent to a video compression driver to stop its internal clock for the timing

of drawing frames.

Parameters DWORD *dwParam1*

Not used.

DWORD *dwParam2*

Not used.

Return Value None.

Comments This message is typically used by hardware which does its own asynchronous decompression, timing, and drawing.

See Also ICM_DRAW, ICM_DRAW_END, ICM_DRAW_START

ICM_DRAW_WINDOW

This message is sent to a video compression driver when the window specifed in the **ICM_DRAW_BEGIN** message has physically moved, or has become totally obscured. This message is used by overlay drivers, so they can draw when the window is obscured or moved.

Parameters DWORD *dwParam1*

Points to a RECT structure containing the destination rectangle. The destination rectangle is specified in screen coordinates. If *dwParam1* points to a empty rectangle drawing should be turned off.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if successful.

Comments This message is only supported by hardware which does its own asynchronous decompression, timing, and drawing.

The rectangle is set empty if the window is totally hidden by other windows. Drivers should turn off overlay hardware when the rectangle is empty.

See Also ICM_DRAW_BEGIN

ICM_GETBUFFERSWANTED

This message is sent to a video compression driver to have the driver return information about how much pre-buffering it wishes to do.

Parameters DWORD *dwParam1*

Specifies a far pointer to a DWORD. The driver uses the DWORD to return the number of samples it needs to get in advance of when they will be presented.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if successful. Otherwise, return ICERR_UNSUPPORTED.

Comments Typically, this message is only used by a driver that uses hardware to render data and wants to ensure hardware pipelines remain full. For example, if a driver controls a video decompression board that can hold ten frames of video, it could return ten for this message. This instructs applications to try and stay exactly ten frames ahead of the frame it currently needs.

ICM_GETDEFAULTKEYFRAMERATE

This message is sent to a video compression driver to request that it return its default (or preferred) key frame spacing.

Parameters DWORD *dwParam1*

Specifies a far pointer to a DWORD used by the driver to return its preferred key frame spacing.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if the driver supports this message. Otherwise, return ICERR_UNSUPPORTED.

ICM_GETDEFAULTQUALITY

This message is sent to a video compression driver to request that the driver return its default quality setting.

Parameters DWORD *dwParam1*

Specifies a far pointer to a DWORD used by the driver to return its default quality.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if the driver supports this message. If not, return

ICERR_UNSUPPORTED.

Comments Quality values range between 0 and 10,000.

See Also ICM_SETQUALITY, ICM_GETQUALITY

ICM_GETINFO

This message is set to a video compression driver to have it return information describing the driver.

Parameters DWORD *dwParam1*

Specifies a far pointer to an **ICINFO** data structure used by the driver to return information.

DWORD *dwParam2*

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

Return Value Return the size of the **ICINFO** data structure, or zero if an error occurs.

Comments Typically, this message is sent by applications that want to display a list of the installed compressors.

The driver should fill in all fields of the **ICINFO** structure except the **szDriver** field.

See Also ICM_ABOUT

ICM_GETQUALITY

This message is sent to a video compression driver to request that it return its current quality setting.

Parameters DWORD *dwParam1*

Specifies a far pointer to a DWORD used by the driver to return the current quality value.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if the driver supports this message. If not, return ICERR_UNSUPPORTED.

Comments Quality values range between 0 and 10,000.

See Also ICM_SETQUALITY, ICM_GETDEFAULTQUALITY

ICM_GETSTATE

This message is sent to a video compression driver to have it fill a block of memory describing the compressor's current configuration.

Parameters DWORD *dwParam1*

Specifies a far pointer to a block of memory to be filled with the current state or NULL. If NULL, return the amount of memory required by the state information.

DWORD *dwParam2*

Specifies the size of the block of memory.

Return Value Return the amount of memory required by the state information.

Comments Client applications send this message with *dwParam1* set to NULL to determine the size of the memory block required for obtaining the state information.

The data structure used to represent state information is driver specific and is defined by the driver.

See Also DRV_CONFIGURE, ICM_SETSTATE

ICM_SETQUALITY

This message is sent to a video compression driver to set the quality level for compression.

Parameters DWORD *dwParam1*

Specifies the new quality value.

DWORD *dwParam2*

Not used.

Return Value Return ICERR_OK if the driver supports this message. If not, return ICERR_UNSUPPORTED.

Comments Quality values range between 0 and 10,000.

See Also ICM_GETQUALITY, ICM_GETDEFAULTQUALITY

ICM_SETSTATE

This message is sent to a video compression driver to set the state of the compressor.

Parameters DWORD *dwParam1*

Specifies a far pointer to a block of memory containing configuration data or NULL. If NULL, the driver should return to its default state.

DWORD *dwParam2*

Specifies the size of the block of memory.

Return Value Return the number of bytes actually used by the compressor. A return value of zero generally indicates an error.

Comments Since the information used by **ICM_SETSTATE** is private and specific to a given compressor, client applications should use this message only to pass information previously returned for the **ICM_GETSTATE** message.

See Also DRV_CONFIGURE, ICM_GETSTATE

Video Compression and Decompression Driver Data Structure Reference

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

ICCOMPRESS

The **ICCOMPRESS** structure is used with the **ICM_COMPRESS** message to specify the parameters used for compression.

```
typedef struct {
   DWORD  dwFlags;
   LPBITMAPINFOHEADER  lpbiOutput;
   LPVOID  lpOutput;
   LPBITMAPINFOHEADER  lpbiInput;
   LPVOID  lpInput;
   LPDWORD  lpckid;
   LPDWORD  lpdwFlags;
   LONG  lFrameNum;
   DWORD  dwFrameSize;
   DWORD  dwQuality;
   LPBITMAPINFOHEADER  lpbiPrev;
   LPVOID  lpInput;
} ICCOMPRESS;
```

Fields The **ICCOMPRESS** structure has the following fields:

dwFlags

Specifies flags used for compression. The following flag is defined.

ICCOMPRESS_KEYFRAME

Treat input data as a keyframe.

lpbiOutput

Specifies a pointer to a **BITMAPINFOHEADER** structure containing the output (compressed) format. The **biSizeImage** field must be filled in with the size of the compressed data.

lpOutput

Specifies a pointer to the buffer where the driver should write the compressed data.

lpbiInput

Specifies a pointer to a **BITMAPINFOHEADER** structure containing the input format.

lpInput

Specifies a pointer to the buffer containing input data.

lpckid

Specifies a pointer to a buffer used to return the chunk ID for data in the AVI file. Device drivers can ignore this field.

lpdwFlags

Specifies a pointer to a buffer used to return flags for the AVI index.

lFrameNum

Specifies the frame number of the frame to compress.

dwFrameSize

Specifies zero, or the desired maximum size (in bytes) to compress this frame to.

dwQuality

Specifies the compression quality.

lpbiPrev

Specifies a pointer to a **BITMAPINFOHEADER** structure containing the format of the previous frame. Normally, this will be the same as the input format.

lpInput

Specifies a pointer to the buffer containing the previous frame.

ICDECOMPRESS

The **ICDECOMPRESS** structure is used with the **ICM_DECOMPRESS** message to specify the parameters for decompressing the data.

```
typedef struct {
   DWORD  dwFlags;
   LPBITMAPINFOHEADER  lpbiInput;
   LPVOID  lpInput;
   LPBITMAPINFOHEADER  lpbiOutput;
   LPVOID  lpOutput;
   DWORD  ckid;
} ICDECOMPRESS;
```

Fields

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

dwFlags

Specifies flags.

The following flags in **dwFlags** specify the operation for this data:

ICDECOMPRESS_HURRYUP
Indicates the data is just buffered and not drawn to the screen. Use this flag for the fastest decompression.

lpbiInput

Specifies a pointer to a **BITMAPINFOHEADER** structure containing the input format.

lpInput

Specifies a pointer to a data buffer containing the input data.

lpbiOutput

Specifies a pointer to a **BITMAPINFOHEADER** structure containing the output format.

lpOutput

Specifies a pointer to a data buffer where the driver should write the decompressed image.

ckid

Specifies the chunk ID from the AVI file.

ICDRAW

The **ICDRAW** structure is used with the **ICM_DRAW** message to specify the parameters for drawing video data to the screen.

```
typedef struct {
   DWORD  dwFlags;
   LPVOID  lpFormat;
   LPVOID  lpData;
   DWORD  cbData;
   LONG  lTime;
} ICDRAW;
```

Fields

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

dwFlags

Specifies the flags from the AVI file index.

ICDRAW_HURRYUP
: Indicates the data is just buffered and not drawn to the screen. Use this flag for the fastest decompression.

ICDRAW_UPDATE
: Indicates the driver should update the screen based on data previously received.

ICDRAW_PREROLL
: Indicates that this frame of video occurs before actual playback should start. For instance, if playback is to begin on frame 10, and frame 0 is the nearest previous keyframe, frames 0 through 9 are sent to the driver with the **ICDRAW_PREROLL** flag set. The driver needs this data so that it can display frame 10 properly, but frames 0 through 9 need not be individually displayed.

lpFormat

Specifies a pointer to a structure containing the data format. For video, this will be a **BITMAPINFOHEADER** structure.

lpData

Specifies the data to be rendered.

cbData

Specifies the number of bytes of data to be rendered.

lTime

Specifies the time in samples that this data should be drawn. For video data this is normally a frame number. See **dwRate** and **dwScale** of the **ICDRAW** structure.

See Also

ICM_DRAW_BEGIN, ICDRAWBEGIN

ICDRAWBEGIN

The **ICDRAWBEGIN** structure is used with the **ICM_DRAW_BEGIN** message to

specify the parameters used to decompress the data.

```
typedef struct {
   DWORD  dwFlags;
   HPALETTE  hpal;
   HWND  hwnd;
   HDC  hdc;
   int  xDst;
   int  yDst;
   int  dxDst;
   int  dyDst;
   LPBITMAPINFOHEADER  lpbi;
   int  xSrc;
   int  ySrc;
   int  dxSrc;
   int  dySrc;
   DWORD  dwRate;
   DWORD  dwScale;
} ICDRAWBEGIN;
```

Fields

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

dwFlags

Specifies any of the following flags:

ICDRAW_QUERY
: Set when an application wants to determine if the device driver can handle the operation. The device driver does not actually perform the operation.

ICDRAW_FULLSCREEN
: Indicates the full screen is used to draw the decompressed data.

ICDRAW_HDC
: Indicates a window or DC is used to draw the decompressed data.

hpal

Specifies a handle of the palette used for drawing.

hwnd

Specifies the handle of the window used for drawing.

hdc

Specifies the handle of the display context used for drawing.

xDst

Specifies the x-position of destination rectangle.

yDst

Specifies the y-position of destination rectangle.

dxDst

Specifies the width of destination rectangle.

dyDst

Specifies the height of destination rectangle.

lpbi

Specifies a pointer to a **BITMAPINFOHEADER** data structure containing the input format.

xSrc

Specifies the x-position of source rectangle.

ySrc

Specifies the y-position of source rectangle.

dxSrc

Specifies the width of source rectangle.

dySrc

Specifies the height of source rectangle.

dwRate

Specifies the decompression rate in an integer format. To obtain the rate in frames-per-second divide this value by the value in *dwScale*.

dwScale

Specifies the value used to scale *dwRate* to frames-per-second.

ICINFO

The **ICINFO** structure is filled by a video compression driver when it receives the **ICM_GETINFO** message.

```
typedef struct {
   DWORD  dwSize;
   DWORD  fccType;
   DWORD  fccHandler;
   DWORD  dwFlags;
   DWORD  dwVersion;
   DWORD  dwVersionICM;
   char  szName[16];
   char  szDescription[128];
   char  szDriver[128];
} ICINFO;
```

Fields

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

dwSize

Should be set to the size of an **ICINFO** structure.

fccType

Specifies a four-character code representing the type of stream being compressed or decompressed. Set this to 'vidc' for video streams.

fccHandler

Specifies a four-character code identifying a specific compressor.

dwFlags

Specifies any flags. The following flags are defined for video compressors (**ICINFO.fccHandler** == 'vidc'):

VIDCF_QUALITY
The driver supports quality.

VIDCF_CRUNCH
The driver supports crunching to a frame size.

VIDCF_TEMPORAL
The driver supports inter-frame compression.

VIDCF_DRAW
The driver supports drawing.

VIDCF_FASTTEMPORAL
The driver can do temporal compression and doesn’t need the previous frame.

dwVersion

Specifies the version number of the driver.

dwVersionICM

Specifies the version of the ICM supported by this driver; it should be set to 1.0 (0x00010000)

szName[16]

Specifies the short name for the compressor. The null-terminated name should be suitable for use in list boxes.

szDescription[128]

Specifies a null-terminated string containing the long name for the compressor.

szDriver[128]

Specifies a null-terminated string for the module that contains the driver. Normally, a driver will not need to fill this out.

ICOPEN

The **ICOPEN** structure is sent to a video compression driver with the **DRV_OPEN** message.

```
typedef struct {
   DWORD  fccType;
   DWORD  fccHandler;
   DWORD  dwVersion;
   DWORD  dwFlags;
} ICOPEN;
```

Fields

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

fccType

Specifies a four-character code representing the type of stream being compressed or decompressed. For video streams, this should be 'vidc'.

fccHandler

Specifies a four-character code identifying a specific compressor.

dwVersion

Specifies the version of the installable driver interface used to open the driver.

dwFlags

Contains flags indicating why the driver is opened. The following flags are defined:

ICMODE_COMPRESS
: The driver is opened to compress data.

ICMODE_DECOMPRESS
: The driver is opened to decompress data.

ICMODE_QUERY
: The driver is opened for informational purposes, rather than for actual compression.

ICMODE_DRAW
: The device driver is opened to decompress data directly to hardware.

Comments

This structure is the same as that passed to video capture drivers when they are opened. This lets a single installable driver to function as either an installable compressor or a video capture device. By examining the **fccType** field of the **ICOPEN** structure, the

driver can determine its function. For example, a **fccType** value of 'vidc' indicates that it is opened as an installable video compressor.