CHAPTER 1

Using the Installable Compression Manager

The Installable Compression Manager (ICM) provides services for applications that want to compress or decompress video image data stored in AVI files. This chapter explains the programming techniques used to access these services. It covers the following topics:

- General information about the ICM and the Video for Windows architecture
- Information on how to compress and decompress video image data from your application
- An alphabetic reference to the ICM functions and data structures

Before reading this chapter, you should be familiar with the video services available with Windows. For information about these Windows topics, see the *Microsoft Windows Programmer's Reference*.

Video Compression and Decompression Header Files

The function prototypes, constants, flags, and data structures applications use to access the ICM services are defined in COMPMAN.H and COMPDDK.H.

ICM Architecture

The ICM is used by the Video for Windows editing tool (VidEdit) and the playback engine (MCIAVI) to handle compression and decompression of image data. ICM is the intermediary between the application and the actual video compression and decompression drivers. It is the video compression/decompression drivers that do the real work of compressing and decompressing individual frames of data.

This chapter covers the ICM and the functions a video editing or playback application uses to communicate with it. For information on the video compression and decompression drivers, see Chapter 10, *Video Compression and Decompression Drivers*.

As the application makes calls to the ICM to compress or decompress data, the ICM translates this to a message to be sent to the appropriate compressor or decompressor which does the work of compressing or decompressing the data. The ICM gets the return from the driver and then returns back to the application.

The ICMAPP sample application illustrates routines that compress data, decompress data, and display a dialog box. You might find the helper functions defined in ICM.C useful in developing your application.

Using ICM Services

In general, an application performs the following tasks to use ICM services:

- Locate, open, or install the appropriate compressor or decompressor
- Configures or obtains configuration information about the compressor or decompressor
- Uses a series of functions to compress, decompress, and (for decompressors with drawing capabilities) draw the data

These tasks are covered in the following sections. The sample application, ICMApp, shows how to use the ICM services to do all of the above functions to compress and decompress images.

Error Returned from the ICM Functions

For most ICM functions, return values of less than zero indicate an error. Your application should check these return values to see if the ICM function encounters an error. To keep the example fragments in this chapter simple, many of them do not check for errors. For more complete examples, see the ICMAPP and ICM examples included with the development kit.

Locating and Opening Compressors and Decompressors

To use ICM, an application must open a compressor or decompressor. If your application does not know about the compressors or decompressors installed on a system, it must find a suitable compressor to open. Once your application finishes with a compressor or decompressor, it closes it to free any resources used for compression or decompression. Your application can use the following functions for finding compressors and decompressors, and opening and closing them:

ICInfo

This function obtains information about compressor or decompressor.

ICOpen

This function opens a compressor or decompressor.

ICClose

This function closes a compressor or decompressor.

ICLocate

This function locates a specific type of compressor or decompressor.

If your application knows the compressor or decompressor it needs, it can open the compressor with the **ICOpen** function. Your application uses the handle returned by this function to identify the opened compressor or decompressor when it uses other ICM functions. The **ICOpen** function has the following syntax:

BOOL ICOpen(*fccType*, *fccHandler*, *wMode*)

The *fccType* and *fccHandler* parameters are four character codes used to describe the type and handler type for the compressor. Compressor and decompressors are identified by two four-character codes. Applications open a specific compressor or decompressor by using the four-character codes for the type and handler. The first four-character code describes the type of the compressor or decompressor. For video compressors and decompressors, this is always 'vidc'. The second four-character code identifies the specific compression handler type. For example, this value is 'msvc' for the Video 1 compressor. Your application can use NULL if it does not know this four-character code.

Note: In an AVI file, the stream header contains information about the stream type and the specific handler for that stream. For video streams, the stream type is 'vidc' and the handler type is the appropriate handler four-character code. As in the previous example, Video 1 compressed streams use 'msvc'.

The *wMode* parameter specifies flags passed to the compressor or decompressor. For **ICOpen**, these flags let the compressor or decompressor know why it is opened and they can prepare for subsequent operation. The following flags are defined:

ICMODE_COMPRESS

Advises a compressor it is opened for compression.

ICMODE_DECOMPRESS

Advises a decompressor it is opened for decompression.

ICMODE_DRAW

Advises a decompressor it is opened to decompress an image and draw it directly to hardware.

ICMODE_QUERY

Advises a compressor or decompressor it is opened to obtain information.

If your application does not know which compressors and deompressors are installed on a system, it can use **ICInfo** to enumerate them. This function has the following syntax:

BOOL ICInfo(*fccType*, *fccHandler*, *lpicinfo*)

The *fccType* parameter specifies a four-character code indicating the type of compressor or decompressor. To enumerate the compressors or decompressors, your application

specifies an integer for *fccHandler*. Your application receives return information for integers between 0 and the number of installed compressors or decompressors of the type specified for *fccType*. The compressor or decompressor returns information about itself in a ICINFO data structure pointed to by *lpicinfo*. The **ICInfo** function returns TRUE if it can locate the specified compressor or decompressor.

The following example enumerates the compressors or decompressors in the system to find one that can handle the format of its images. (The example uses **ICCompressQuery** and **ICDecompressQuery** to determine if a compressor or decompressor supports the image format. The use of these functions is described in "Compressing Image Data" and "Decompressing Image Data.")

```
for (i=0; ICInfo(p->fccType, i, &p->icinfo); i++)
{
   hic = ICOpen(p->icinfo.fccType, p->icinfo.fccHandler, ICMODE_QUERY);

   if (hic)
   {
      // skip this compressor if it can't handle the specifed format
      if (p->fccType == ICTYPE_VIDEO &&
         p->pvIn != NULL &&
         ICCompressQuery(hic, p->pvIn, NULL) != ICERR_OK &&
         ICDecompressQuery(hic, p->pvIn, NULL) != ICERR_OK)
      {
         ICClose(hic);
         continue;
      }

      // find out the compressor name.
      ICGetInfo(hic, &p->icinfo, sizeof(p->icinfo));

      // stuff it into the combo box
      n = ComboBox_AddString(hwndC,p->icinfo.szDescription);
      ComboBox_SetItemData(hwndC, n, hic);
   }
}
```

Applications can use **ICLocate** to find a compressor or decompressor of a specific type, and to obtain a handle to it for use in other ICM functions. The **ICLocate** function has the following syntax:

HIC ICLocate (*fccType*, *fccHandler*, *lpbiIn*, *lpbiOut*, *wFlags*)

The *fccType* and *fccHandler* parameters are four-character codes used to describe the type and handler type for the compressor. Your application can specify NULL for *fccHandler* if it does not know the handler type or if it can use any handler type.

The *lpbiIn* parameter contains a pointer to a BITMAPINFOHEADER structure describing the input format the compressor or decompressor will handle. The *lpbiOut* parameter contains a pointer to a BITMAPINFOHEADER structure describing the output format desired by the application. If your application does not care what output format is returned by a compressor or decompressor, you can set *lpbiOut* to NULL. The *wFlags* parameter indicates the type of operation you want the driver to do: compress, decompress or directly decompress and draw.

For example, the following fragment tries to find a compressor that can compress an 8-bit

per pixel bitmap:

```
BITMAPINFOHEADER    bih;
HIC                 hIC

// inialize the Bitmap structure
bih.biSize = sizeof(BITMAPINFOHEADER);
bih.biWidth = bih.biHeight = 0;
bih.biPlanes = 1;
bih.biCompression = BI_RGB;      // standard RGB bitmap
bih.biBitcount = 8;              // 8bpp format
bih.biSizeImage = 0;
bih.biXPelsPerMeter = bih.biYPelsPerMeter = 0;
bih.biClrUsed = bih.biClrImportant = 256;

hIC = ICLocate (ICTYPE_VIDEO, 0L,
                (LPBITMAPINFOHEADER)&bih, NULL, ICMODE_COMPRESS);
```

The following fragment tries to locate a specific compressor to compress the 8-bit RGB format to an 8-bit RLE format:

```
BITMAPINFOHEADER    bihIn, bihOut;
HIC                 hIC

// initialize the Bitmap structure
bihIn.biSize = bihOut.biSize = sizeof(BITMAPINFOHEADER);
bihIn.biWidth = bihIn.biHeight = bihOut.biWidth = bihOut.biHeight = 0;
bihIn.biPlanes = bihOut.biPlanes= 1;
bihIn.biCompression = BI_RGB;              //standard RGB bitmap for input
bihOut.biCompression = BI_RLE8;            // 8-bit RLE for output format
bihIn.biBitcount = bihOut.biBitCount = 8;  // 8bpp format
bihIn.biSizeImage = bihOut.biSizeImage = 0;
bihIn.biXPelsPerMeter = bih.biYPelsPerMeter =
   bihOut.biXPelsPerMeter = bihOut.biYPelsPerMeter = 0;
bihIn.biClrUsed = bih.biClrImportant =
   bihOut.biClrUsed = bihOut.biClrImportant = 256;

hIC = ICLocate (ICTYPE_VIDEO, 0L,
      (LPBITMAPINFOHEADER)&bihIn,
      (LPBITMAPINFOHEADER)&bihOut, ICMODE_COMPRESS);
```

Installing and Removing Compressors and Decompressors

There are three methods of installing compressors and decompressors. They might be installed during the set-up of Video for Windows or other software relating to Video for Windows. Users can also install compressors and decompressors with the Drivers option of the Control Panel. Applications can also install custom compressors and decompressors functions required for their operation. Most applications will not need to install or remove compressors or decompressors. Your application can use the following functions for installing and removing compressors and decompressors:

ICInstall

This function installs compressor or decompressor.

ICRemove

This function removes an installed compressor or decompressor.

Compressors and decompressor are usually installed by a setup program or by the user with the Drivers option of the Control Panel. An application might, however, install a compressor directly or install a function as a compressor. In these cases, the application uses **ICInstall** and **ICRemove**.

The **ICInstall** function creates a new entry in the SYSTEM.INI for a compressor or decompressor. After installation, the compressor or decompressor must still be opened. The **ICInstall** function has the following syntax:

BOOL ICInstall (*fccType*, *fccHandler*, *lParam*, *szDesc*, *wFlags***)**

The *fccType* and *fccHandler* parameters specify four-character codes describing the compressor type and handler type. Flags set in *wFlags* specify the meaning of *lParam*. The following flags are defined:

ICINSTALL_DRIVER

Indicates *lParam* points a null-terminated string containing the name of a compressor or decompressor.

ICINSTALL_FUNCTION

Indicates *lParam* points to a compressor function.

If you are installing a driver, *lParam* specifies the name of the driver. If you are installing a function as a compressor or decompressor, use *lParam* to specify a far pointer to your function. This function should be structured like the **DriverProc** entry point function used by installable drivers. For more information on the **DriverProc** entry point function, see Chapter 10, “Video Compression and Decompression Drivers.”

Use the *szDesc* parameter for a descriptive name for the compressor or decompressor. This information is not used and your application does not have to supply a name.

For example, the following fragment installs the ICSample driver:

```
result = ICInstall ( ICTYPE_VIDEO, mmioFOURCC('s','a','m','p'),
        (LPARAM)(LPSTR)"icsample.drv", "Sample Codec Driver", ICINSTALL_DRIVER)
```

The following fragment shows how an application would install a function as a compressor or decompressor.

```
// This function looks like a DriverProc entry point
LRESULT MyCodecFunction(DWORD dwID, HDRVR hDriver, UINT uiMessage,
                        LPARAM lParam1, LPARAM lParam2);

// This function installs the MyCodecFunction as a compressor
result = ICInstall ( ICTYPE_VIDEO, mmioFOURCC('s','a','m','p'),
                              (LPARAM)(FARPROC)&MyCodecFunction, NULL, ICINSTALL_FUNCTION);
```

Usually the Drivers option of the Control Panel is used to remove a compressor or decompressor. Applications installing a function as a compressor or decompressor must remove the function before the application terminates so other applications do not try to use the function. Applications can use **ICRemove** to remove the function installed. The **ICRemove** function has the following syntax:

BOOL ICRemove (*fccType*, *fccHandler*, *wFlags***)**

The *fccType* and *fccHandler* parameters specify four-character codes describing the compressor type and handler type. The *wFlags* parameter is not used.

Configuring Compressors and Decompressors

Applications can configure the compressor or have the compressor display a dialog box to let the user to do the configuration. The following functions are available for these operations:

ICQueryConfigure

Determines if the compressor or decompressor supports a configuration dialog box.

ICConfigure

Displays the configuration dialog box of the compressor or decompressor.

ICGetStateSize

Determines the size of the state data for the compressor or decompressor.

ICGetState

Obtains the state data from the compressor or decompressor.

ICSetState

Sends the state data to the compressor or decompressor.

If practical, your application should let the user configure the compressor with the compressor's configuration dialog box. Typically, this makes your application independent of the compressor and you do not need to consider all the options available to a compressor.

Your application uses **ICQueryConfigure** to determine if a compressor can display a configuration dialog box. If the compressor can display a configuration dialog box, your

application uses **ICConfigure** to display it. Both of these functions use the handle your application obtained when it located the compressor. The **ICConfigure** function also requires a handle to the parent window. Your application can use the following fragment to test for support of the configuration dialog box and display it:

```
if (ICQueryConfigure(hIC)){

   // If compressor handles a configuration dialog box, display it
   // using our app window as the parent window.
   ICConfigure(hIC, hwndApp);

}
```

Your application might also directly get and set the state information for a compressor. If your application creates or modifies the state data, it must know the actual layout of the compressor data before restoring a compressor state. Alternatively, if your application obtains state data from a compressor and uses it to restore the state in a subsequent session, it must make sure that it only restores state information obtained from the compressor it is currently using. The following fragments show how to obtain the state data:

```
if (size > 0) {

   dwStateSize = ICGetStateSize(hIC);        // get size of buffer required
   h = GlobalAlloc(GHND, dwStateSize);       // allocate data buffer
   lpData = GlobalLock(h);                   // lock data buffer
   ICGetState(hIC, (LPVOID)lpData, dwStateSize); // get the state data

   // Store the state data as required
}
```

The following fragments show how to restore state data:

```
ICSetState(hIC, (LPVOID)lpData, dwStateSize); // set the new state data
GlobalUnlock(h);
```

Getting Information about Compressors and Decompressors

The following functions can be used to get information about a compressor or decompressor:

ICGetInfo

Obtains general information about the compressor or decompressor.

ICGetDefaultKeyFrameRate

Determines the default key frame rate of a compressor or decompressor.

ICGetDisplayFormat

Determines the 'best' format a compressor or decompressor has for displaying on the screen.

ICGetDefaultQuality

Determines the default quality value of a compressor or decompressor.

To obtain information about a compressor or decompressor, your application can use **ICGetInfo**. This function fills an ICINFO structure with information about the compressor or decompressor. Your application must allocate the memory for the ICINFO structure and pass a pointer to it in **ICGetInfo**. The ICINFO structure has the following definition:

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

The **dwSize** field contains the size of the ICINFO structure.

The **fccType** field contains the four-character code 'vidc' for image compression.

The **fccHandler** field identifies the compressor or decompressor with its four-character code.

The **dwVersion** field contains the version number of the compression driver.

The **dwVersionICM** field will contain the ICM version number supported by the compressor or decompressor. This is 1.0 (0x00010000) if the compressor or decompressor is written for Video for Windows 1.0.

The **szName** field contains the short name of the compressor or decompressor. The name is used in list-boxes for choosing compression methods.

The **szDescription** field contains the long description for the compressor or decompressor.

The **szDriver** field contains the actual module name that contains the compressor or decompressor.

The **dwFlags** field contains flags indicating capabilities of the compressor or decompressor. The following flags are defined:

VIDCF_QUALITY

Indicates the compressor or decompressor supports quality levels.

VIDCF_CRUNCH

Indicates a compressor supports compressing to an arbitrary frame size.

VIDCF_TEMPORAL

Indicates the compressor or decompressor supports inter-frame compression.

VIDCF_DRAW

Indicates decompressor can draw to hardware. (These decompressors support the ICDraw functions.)

VIDCF_FASTTEMPORAL

Indicates a compressor can do temporal compression but it doesn't need previous frame.

Unless your application is looking for a particular compressor or decompressor, the flags give your application the most useful information. Your application can check the flags to determine the capabilities of the compressor or decompressor. For example, if quality is supported, your application might enable a quality selection control in a compression dialog box.

The following fragment shows how to obtain the ICINFO information from a compressor or decompressor:

```
ICINFO          ICInfo;

ICGetInfo(hIC, &ICInfo, sizeof(ICInfo));
```

The **ICGetDefaultKeyFrameRate** and **ICGetDefaultQuality** functions let your application determine the default key frame rate and default quality value. Both of these functions require only the handle to the compressor or decompressor. The following fragment uses both functions to obtain the default values:

```
DWORD    dwKeyFrameRate, dwQuality;

dwKeyFrameRate = ICGetDefaultKeyFrameRate(hIC);
dwQuality = ICGetDefaultQuality(hIC);
```

Your application can use the following functions to display the about dialog box of a compressor or decompressor:

ICQueryAbout

Determines if a compressor or decompressor supports an about dialog box.

ICAbout

Displays the about dialog box of a compressor or decompressor.

The **ICQueryAbout** function lets your application determine if a compressor or decompressor can display the about dialog box. The **ICAbout** function actually displays the dialog box. The following examples uses these two functions:

```
if ( ICQueryAbout(hIC)){

   // If the compressor has an about dialog box, show it
   ICAbout(hIC, hwndApp);
}
```

Compressing Image Data

Your application uses a series of functions to coordinate compressing video data. The coordination involves the following activities:

- Specifying the input format and determining the compression format
- Preparing the compressor for compression
- Compressing the video
- Ending compression

Your application uses the following functions for these activities:

ICCompress

Compress data.

ICCompressBegin

Prepare compressor driver for compressing data.

ICCompressEnd

Tell the compressor driver to end compression.

ICCompressGetFormat

Determine the output format of a compressor.

ICCompressGetFormatSize

Get the size of the output format data.

ICCompressGetSize

Get the size of the compressed data.

ICCompressQuery

Deterimine if a compressor can compress a specific format.

Specifying the Input Format and Determining the Compression Format

When your application wants to compress data and the output format is not important, it must first locate a compressor that can handle the input format. When the output format is not important to your application, it can use **ICCompressGetFormat** to have the compressor suggest a format. If the compressor can produce multiple formats, it returns the format 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 **ICCompressGetFormat** function has the following syntax:

LRESULT ICCompressGetFormat(*hic*, *lpbiInput*, *lpbiOutput*)

The *hic* parameter specifies the compressor handle. The *lpbiInput* parameter specifies a far pointer to a BITMAPINFO structure indicating the format of the input data. The *lpbiOutput* parameter specifies a far pointer to a buffer used to return the output format suggested by the compressor. Your application can determine the size of the buffer needed for the buffer with **ICCompressGetFormatSize**.

Your application can use the output format data as the 'strf' chunk in the AVI RIFF file. This data starts out like a BITMAPINFOHEADER data structure. The compressor can include any additional information required to decompress the file after this information. A color table (if used) follows this information. If the compressor has format data following the BITMAPINFOHEADER structure, it updates the **biSize** field to specify the number of bytes used by the structure and additional data.

The following example fragment shows how an application can determine the output format that a compressor wants to use.

```
LPBITMAPINFOHEADER   lpbiIn, lpbiOut;

// *lpbiIn must be initialized to the input format

dwFormatSize = ICCompressGetFormatSize(hIC, lpbiIn); // get output buffer size
h = GlobalAlloc(GHND, dwFormatSize);            // allocate format buffer
lpbiOut = (LPBITMAPINFOHEADER)GlobalLock(h);     // lock format buffer
ICCompressGetFormat(hIC, lpbiIn, lpbiOut);       // fill the format information
```

If your application requires a specific output format, it should use **ICCompressQuery** to interrogate a compressor to determine if it supports the output format your application suggests. This function has the following syntax:

LRESULT ICCompressQuery(*hic*, *lpbiInput*, *lpbiOutput*)

The *hic* parameter specifies the compressor handle. Your application typically obtains this with **ICLocate** or **ICOpen.** The *lpbiInput* and *lpbiOutput* parameters specify far pointers to the data structures defining the input and output formats your application prefers. If the compressor can handle both formats it returns ICERR_OK. If it cannot handle the formats, it returns ICERR_BADFORMAT. If the compressor returns ICERR_BADFORMAT and the output format is critical to your application, your

application will have to find an alternate compressor. If an alternate output format is satisfactory, your application might choose to use **ICCompressQuery** with the alternate formats to determine if the compressor can handle them. Or your application can use **ICCompressGetFormat** to have the compressor suggest the output format.

If your application specifies NULL for the *lpbiOutput* parameter of **ICCompressQuery**, the compressor will select the output format. Typically, your application specifies NULL when it only wants to know if the compressor can handle the input format. The output format information is not returned to your application.

The following fragment uses **ICCompressQuery** to determine if a compressor can handle both the input and output format:

```
LPBITMAPINFOHEADER   lpbiIn, lpbiOut;

// Both *lpbiIn & *lpbiOut must be initialized to the respective formats
if (ICCompressQuery(hIC, lpbiIn, lpbiOut) == ICERR_OK){

        // format is supported - use the compressor

}
```

Your application will also need the size of the data returned from the compressor after compression is complete. Use **ICCompressGetSize** to obtain the worst case (largest) buffer required by the compressor. The number of bytes returned should be used to allocate a buffer used for subsequent compression of images. The following example determines the buffer size and allocates a buffer of that size:

```
// find the worst-case buffer size
dwCompressBufferSize = ICCompressGetSize(hIC, lpbiIn, lpbiOut);

// allocate a buffer and get lpOutput to point to it
h = GlobalAlloc(GHND, dwCompressBufferSize);
lpOutput = (LPVOID)GlobalLock(h);
```

Initialization for the Compression Sequence

Once your application selects a compressor that handles the input and output formats it needs, it can prepare the compressor to start compressing data. The **ICCompressBegin** function initializes the compressor. This function requires the compressor handle and the input and output format. It returns ICERR_OK if it initializes properly for the specified formats. If the compressor cannot handle the formats, or if they are incorrect, it returns the error ICERR_BADFORMAT.

Compressing the Video

The **ICCompress** function does the actual compression. Your application must use this function repeatedly until all the frames are compressed. This function has the following syntax:

LRESULT ICCompress(*hic*, *dwFlags*, *lpbiOutput*, *lpData*, *lpbiInput*, *lpBits*, *lpckid*, *lpdwFlags*, *lFrameNum*, *dwFrameSize*, *dwQuality*, *lpbiPrev*, *lpPrev*)

The *hic* parameter specifies the handle to the compressor.

The *dwFlags* parameter specifies any applicable flags for the compression. Your application can use ICM_COMPRESS_KEYFRAME to have the compressor make the frame a key frame. (A key frame is one that does not require data from a previous frame for decompression.) When this flag is set, compressors use this image as the initial one in a sequence.

The *lpbiInput* and *lpBits* parameters specify far pointers to the data structure defining the input format and the location of the input buffer. Similarly, the *lpbiOutput* and *lpData* parameters specify far pointers to the data structure defining the output format and the location of the buffer for the output data. Your application must allocate the memory for these buffers. When control returns to your application, it typically stores the compressed data in *lpbiOutput* and *lpData* in a subsequent operation. If your application needs to move the compressed data, it can find the size used for the data in the **biSizeImage** field in the BITMAPINFO structure specified for *lpbiOutput*.

The *lpckid* and *lpdwFlags* are used for AVI file data returned by the compressor. The *lpckid* specifies a far pointer to a **DWORD** used to hold a chunk ID for data in the AVI file. The *lpdwFlags s*pecifies a far pointer to a **DWORD** holding the return flags used in the AVI index. The compressor will set this flag to AVIIF_KEYFRAME to correspond to the ICM_COMPRESS_KEYFRAME flag. The AVIIF_KEYFRAME flag marks the key-frames in the AVI file. If your application creates AVI files, it should save the information returned for these parameters in the file.

The *lFrameNum* parameter specifies the frame number. Your application provides and, if necessary, increments this information. Compressors use this value to check if frames are being sent out of order when they are doing fast temporal compression. If your application has a frame recompressed, it should use the same frame number used when the frame was first compressed. If your application compresses a still frame image, it can specify zero for *lFrameNum*.

The *dwFrameSize* parameter specifies the requested frame size in bytes. If set to zero, the compressor chooses the frame size. If set to a non-zero value, the compressor tries to compress the frame to within the specified size. To obtain the size goal, the compressor might have sacrificed image quality (or made some other trade-off). Compressors recognize the frame size value only if they return the VIDCF_CRUNCH flag for **ICGetInfo**.

The *dwQuality* parameter specifies the requested quality value for the frame. Compressors support this only if they set the VIDCF_QUALITY flag for **ICGetInfo.**

The *lpbiPrev* and *lpPrev* parameters specify far pointers to the data structure defining the format and the location of the previous uncompressed image. Compressors use this data if they perform temporal compression (that is, they need the previous frame to compress the

current frame). Compressors need this information only if they return the VIDCF_TEMPORAL flag. Compressors returning the VIDCF_FASTTEMPORAL flag can perform temporal compression without the previous frame.

The **ICCompress** function returns ICERR_OK if successful. Otherwise, it returns an error code.

After your application has compressed its data, it uses **ICCompressEnd** to notify the compressor that it has finished. To restart compression after using this function, your application must re-initialize the compressor with **ICCompressBegin.**

The following fragment compresses image data for use in an AVI file. It assumes the compressor does not support VIDCF_CRUNCH or VIDCF_TEMPORAL flags but it does support VIDCF_QUALITY.

```
DWORD    dwCkID;
DWORD    dwCompFlags;
DWORD    dwQuality;
LONG     lNumFrames, lFrameNum;

// assume dwNumFrames is initialized to the total number of frames
// assume dwQuality holds the proper quality value (0-10000)
// assume lpbiOut, lpOut, lpbiIn and lpIn are all initialized properly.

if (ICCompressBegin(hIC, lpbiIn, lpbiOut) == ICERR_OK){

   // If o.k. to start, compress each frame
   for ( lFrameNum = 0; lFrameNum < lNumFrames; lFrameNum++){

      if (ICCompress(hIC, 0, lpbiOut, lpOut, lpbiIn, lpIn,
         &dwCkID, &dwCompFlags, lFrameNum,
         0, dwQuality, NULL, NULL) == ICERR_OK){


         // Write compressed data the AVI file.
         .
         .
         .
         // set lpIn to be the next frame in the sequence

      } else {

         // handle compressor error

      }

   }


   ICCompressEnd(hIC);    // terminate compression

} else {
```

```
    // handle error

}
```

Decompressing Image Data

Similar to compressing data, your application uses a series of functions to control the decompressor used to decompress the video data. Decompressing data involves the following activities:

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

Your application uses the following functions for these activities:

ICDecompress

Decompress data.

ICDecompressBegin

Prepare decompressor for decompressing data.

ICDecompressEnd

Tell decompressor to end decompression.

ICDecompressGetFormat

Determine the output format of a decompressor.

ICDecompressGetFormatSize

Get the size of the output data format.

ICDecompressGetPalette

Get the palette for the output format of a decompressor.

ICDecompressQuery

Determine if a decompressor can decompress a specific format.

Decompression is handled very much like compression except that the input format is a compressed format and the output is a displayable format. The input format for decompression is usually obtained from the video stream header in the AVI file. After determining the input format, your application can use **ICLocate** or **ICOpen** to find a decompressor that can handle it.

Specifying the Input Format and Determining the Decompression Format

Because your application allocates the memory required for decompression, it needs to determine the maximum memory the decompressor can require for the output format. The **ICDecompressGetFormatSize** function obtains the number of bytes the decompressor uses. This function has the following syntax:

DWORD ICDecompressGetFormatSize(*hic*, *lpbi***)**

The *hic* parameter specifies a handle to a decompressor. The *lpbi* specifies a far pointer to a BITMAPINFO structure indicating the format of the input data.

If your application wants the decompressor to suggest a format, it can use **ICDecompressGetFormat** to obtain the format. This function has the following syntax:

DWORD ICDecompressGetFormat(*hic*, *lpbiInput*, *lpbiOutput*)

Like **ICDecompressGetFormatSize**, the *hic* and *lpbiInput* parameters specify a handle to the decompressor and a far pointer to the structure indicating the format of the input data. The decompressor returns its suggested format in the BITMAPINFO structure pointed to by *lpbiInput.* Your application should check that the decompressor returns ICERR_OK for the return value before accessing the *lpbiOutput* information. If the decompressor cannot handle the input format, it returns ICERR_BADFORMAT. The following fragment shows how an application can use **ICDecompressGetFormat**:

```
LPBITMAPINFOHEADER   lpbiIn, lpbiOut;

// assume *lpbiIn points to the input (compressed) format

dwFormatSize = ICDecompressGetFormatSize(hIC, lpbiIn); // get output
                                        // buffer size
h = GlobalAlloc(GHND, dwFormatSize);         // allocate format buffer
lpbiOut = (LPBITMAPINFOHEADER)GlobalLock(h); // lock format buffer
ICDecompressGetFormat(hIC, lpbiIn, lpbiOut); // fill the format information
```

If your application needs a specific output format, in can use **ICDecompressQuery** to determine if the decompressor can handle both the input and output format. This function uses the same parameters as **ICDecompressGetFormat** except that your application sets *lpbiOutput* to point at the structure defining the desired output format. If your application is just determining if the decompressor can handle the input format, it can specify NULL for *lpbiOutput*. The following fragment shows how an application can use this function:

```
LPBITMAPINFOHEADER   lpbiIn, lpbiOut;

// assume both *lpbiIn & *lpbiOut are initialized to the respective formats
if (ICDecompressQuery( hIC, lpbiIn, lpbiOut) == ICERR_OK){

   // format is supported - use the decompressor

}
```

If your application creates its own format, it must also obtain a palette for the bitmap.

(Most applications use standard formats and do not need to obtain a palette.) Your application can obtain the palette with **ICDecompressGetPalette**. This function has the following syntax:

DWORD ICDecompressGetPalette(*hic*, *lpbiInput*, *lpbiOutput*)

Like the other functions, *hic* and *lpbiInput* specify a handle to a decompressor and point to a **BITMAPINFO** structure indicating the format of the input data. The *lpbiOutput* parameter points to a **BITMAPINFO** structure used to return the color table. The space reserved for the color table must have an entire 256 color palette table reserved at the end of the structure. The following fragment shows how to get the palette information:

```
ICDecompressGetPalette(hIC, lpbiIn, lpbiOut);

// move up to the palette
lpPalette = (LPBYTE)lpbiOut + lpbi->biSize;
```

Initialization for the Decompression Sequence

Once your application selects a decompressor that handles the input and output formats it needs, it can prepare the decompressor to start decompressing data. The **ICDecompressBegin** function initializes the compressor. This function requires the compressor handle and the input and output format. It returns ICERR_OK if it initializes properly for the specified formats. If the compressor cannot handle the formats, or if they are incorrect, it returns the error ICERR_BADFORMAT.

Decompressing the Video

The **ICDecompress** function does the actual decompression. Your application must use this function repeatedly until all the frames are decompressed. This function has the following syntax:

DWORD ICDecompress(*hic*, *dwFlags*, *lpbiFormat*, *lpData*, *lpbi*, *lpBits*)
The *hic* parameter specifies the handle to the decompressor.

The *dwFlags* parameter specifies any applicable flags for decompression. If your video presentation is starting to lag other components (such as audio), your application can use ICM_DECOMPRESS_HURRYUP to have the decompressor decompress at a faster rate. To speed up decompression, a decompressor might extract only the information it needs to decompress the next frame and not fully decompress the current frame. Thus, when your application uses this flag, it should not try to draw the decompressed data.

The *lpbiFormat* and *lpData* parameters specify far pointers to the data structure defining the input format and the location of the input buffer. Similarly, the *lpbi* and *lpBits* parameters specify far pointers to the data structure defining the output format and the location of the buffer for the output data. Your application must allocate the memory for these buffers. When control returns to your application, it will use the information in *lpbi* and *lpBits* for subsequent processing of the decompressed data.

The **ICDecompress** function returns ICERR_OK if successful. Otherwise, it returns an

error code.

After your application has decompressed its data, it uses **ICDecompressEnd** to notify the decompressor that it has finished. To restart decompression after using this function, your application must re-initialize the decompressor with **ICDecompressBegin**. The following fragment shows how an application can initialize a decompressor, decompress a frame sequence, and terminate decompression:

```
LPBITMAPINFOHEADER  lbpiIn, lpbiOut;
LPVOID              lpIn, lpOut;
LONG                lNumFrames, lFrameNum;

// assume lpbiIn and lpbiOut are initialized to the input and output format
// and lpIn and lpOut are pointing to the data buffers.

if (ICDecompressBegin(hIC, lpbiIn, lpbiOut) == ICERR_OK){

   for (lFrameNum = 0; lFrameNum < lNumFrames, lFrameNum++){

      if (ICDecompress(hIC, 0, lpbiIn, lpIn, lpbiOut, lpOut) == ICERR_OK){

         // frame decompressed OK so we can process it as required


      } else {

         // handle decompression error

      }
   }

   ICDecompressEnd(hIC);

} else {

   // handle error for decompression initialization

}
```

Using Hardware Drawing Capabilities

Some decompressors have the ability to draw directly to video hardware as they decompress video frames. These decompressors return the VIDCF_DRAW flag in response to **ICGetInfo**. When using this type of decompressor, your application does not have to handle the decompressed data. It lets the decompressor retain the decompressed data for drawing. The following functions are used to for decompressing and drawing with decompressors that have drawing capabilities:

ICDrawBegin

This function prepares a decompressor for drawing.

ICDrawEnd

This function stops a decompressor’s drawing operations.

ICDrawFlush

This function flushes the buffers in the decompressor.

ICDrawQuery

This function determines if the decompressor can render data in a specific format.

ICDrawStart

This function starts the internal clock a decompressor uses for drawing.

ICDrawStop

This function stops the internal clock a decompressor uses for drawing.

ICGetBuffersWanted

This function determines the pre-buffering requirements of a compressor.

If your application uses a decompressor with drawing capabilities, it must handle the following activities:

- find a decompressor that can decompress and draw a bitmap with the input format specified
- prepare for decompression
- decompress data
- terminate the decompression process

Specifying the Input Format

Since your application no longer needs to draw the final data, it does not need to be concerned with the output format. However, it must make sure the decompressor can draw the input format. Your application can use **ICDrawQuery** to determine if a decompressor can handle the input format. While this function can determine if a deompressor can handle the format, it does not determine if the a decompressor has all the capablities needed to draw a bitmap. If your application is uncertain if the decompressor can render the bitmap as required, use this function with **ICDrawBegin** The following section describes **ICDrawBegin**. The following fragment shows how to check the input format with **ICDrawQuery**:

```
// lpbiIn points to BITMAPINFOHEADER structure indicating the input format

if (ICDrawQuery(hIC, lpbiIn) == ICERR_OK){

   // decompressor recognizes the input format

} else {

   // we need a different decompressor

}
```

Preparing to Decompress Video

The **ICDrawBegin** function initializes a decompressor and it informs the decompressor about the destination of drawing. The **ICDrawBegin** function has the following syntax:

DWORD ICDrawBegin(*hIC*, *dwFlags*, *hPal*, *hwnd*, *hdc*, *xDst*, *yDst*, *dxDst*, *dyDst*,
lpbi, *xSrc*, *yScr*, *dxSrc*, *dySrc*, *dwRate*, *dwScale***)**

The *hIC* parameter contains the handle to the decompressor. The *dwFlags* parameter specifies any applicable flags. The following flags are defined for this function:

ICDRAW_QUERY

Use to determine if the decompressor can handle the decompression. The decompressor does not draw when this flag is used.

ICDRAW_FULLSCREEN

Indicates that the decompressor will draw to the full screen rather than to a window.

ICDRAW_HDC

Indicates that the decompressor will use a window and display context for drawing.

The *hPal* parameter specifies a handle to the palette used for drawing. Decompressor ignore this information and your application can set it to null.

The *hwnd* and *hdc* parameters define the window and display context used for drawing. Your application must set these values if it uses the ICDRAW_HDC flag.

The *xDst*, *yDst*, *dxDst* and *dyDst* parameters define the destination rectangle used for drawing. Specify the destination rectangle values relative to the current window or display context. Your application should set these parameters to the desired destination rectangle if it uses ICDRAW_HDC. It can set them to zero if it uses the ICDRAW_FULLSCREEN flag.

The *xSrc*, *ySrc*, *dxSrc*, and *dySrc* parameters specify the source rectangle used for clipping

the frames of the image. The decompressor will stretch the rectangle specified as the source into the rectangle specified by the destination when drawing.

The *lpbi* parameter should contain a pointer to the BITMAPINFO structure for the input format. Your application uses the *dwRate* and *dwScale* parameters to specify the decompression rate. The integer value specified for *dwRate* divided by the integer value specified for *dwScale* defines the play rate in frames per second. This is used by the decompressor when it is responsible for timing of frames on playback.

The following fragment shows the initialization sequence to have the decompressor draw full screen:

```
// assume lpbiIn has the input format, dwRate has the data rate
if (ICDrawBegin(hIC, ICDRAW_QUERY|ICDRAW_FULLSCREEN, NULL, NULL,
      NULL, 0, 0, 0, 0, lpbiIn, 0, 0, 0, 0, dwRate, dwScale) == ICERR_OK){

   // decompressor supports this drawing so set up to draw.
   ICDrawBegin(hIC, ICDRAW_FULLSCREEN, hPal, NULL, NULL, 0, 0, 0, 0, lpbiIn,
      0, 0, lbpi->biWidth, lpbi->biHeight, dwRate, dwScale);

   // we're ready to start decompressing and drawing frames now

   // drawing done so terminate
   ICDrawEnd(hIC);
} else {

   // do drawing myself

}
```

Some decompressors buffer the compressed data for more efficient operation. Your application can use **ICGetBuffersWanted** to determine how many data frames it should send to the decompressor before it has the decompressor draw them.

Decompressing the Video

The **ICDraw** function has the decompressor do the actual decompression. This function has the following syntax:

LRESULT ICDraw(*hIC*, *dwFlags*, *lpFormat*, *lpData*, *cbData*, *lTime***)**

The *hIC* is the handle to the decompressor. The *dwFlags* are flags set by the application and used by the decompressor. These flags can be:

ICDRAW_HURRYUP

Tell the decompressor to decompress at a faster rate.

ICDRAW_UPDATE

Tell the decompressor to update the screen based on the last data received. In this case the *lpData* parameter should be NULL.

The *lpFormat* parameter specifies a pointer to the format of the input data. The *lpData* parameter contains the actual data to be decompressed and later drawn.

The *cbData* parameter specifies the number of bytes in *lpData*.

The *lTime* parameter specifies the time to draw this frame. The decompressor divides this integer by the time scale specified with **ICDrawBegin** obtain the actual time. Time for the ICDraw functions is relative to **ICDrawStart**. (That is, **ICDrawStart** sets the clock to zero.) For example, if your applications specifies 1000 for the time scale and 75 for *lTime*, the decompressor draws the frame 75 milliseconds into the sequence.

The decompressor starts decompressing data in response to **ICDraw**, however, it does not start drawing data until your application calls **ICDrawStart**. (Your application should not use **ICDrawStart** until it has sent the number of frames the decompressor returned for **ICGetBuffersWanted.)** When your application uses **ICDrawStart**, the decompressor begins to draw the frames at the rate specified by *dwRate* specified with the **ICDrawBegin**. Drawing continues until your application stops the decompressor drawing clock with **ICDrawStop**. The following fragment uses the **ICDraw** functions:

```
DWORD           dwNumBuffers;

// find out how many buffers need filling before drawing starts
ICGetBuffersWanted(hIC, &dwNumBuffers);

for (dw = 0; dw < dwNumBuffers; dw++){

        ICDraw(hIC, 0, lpFormat, lpData, cbData, dw); // fill the pipeline

    // Point lpFormat and lpData to next format and data buffer

}

ICDrawStart(hIC); // start the clock

while (fPlaying){

        ICDraw(hIC, 0, lpFormat, lpData, chData, dw); // fill more buffers

    // Point lpFormat and lpData to next format and data buffer, update dw

}

ICDrawStop(hIC);  // when done stop drawing and decompressing
ICDrawFlush(hIC); // flush any existing buffers
ICDrawEnd(hIC);   // end decompression
```

Controlling Drawing Parameters

The following functions provide more control over decompressors that can draw the decompressed data:

ICDrawGetTime

This function obtains the current time from the decompressor.

ICDrawRealize

This function has the decompressor realize the palette used for drawing.

ICDrawSetTime

This function sets the value of the internal clock for the decompressor.

ICDrawWindow

This function has the decompressor redraw the window.

If your application wants to monitor or change the clock of the decompressor, it can use **ICDrawGetTime** and **ICDrawSetTime**. If your application wants to change the playback position while the decompressor is drawing, it can use **ICDrawWindow** for repositioning the decompressor. If the playback window gets a palette realize message, your application must call **ICDrawRealize** to have the decompressor realize the palette again for playback.

Video Compression and Decompression Application Reference

This section is an alphabetic reference to the functions and data structures provided by ICM for applications using video compression and decompression services. There are separate sections for functions and data structures. The COMPMAN.H and COMPDDK.H files define the functions and data structures.

Video Compression and Decompression Function Reference

Applications use the following functions for compressing video data:

ICCompress

This function compresses a single video image.

ICCompressBegin

This functions prepares a compressor for compressing data.

ICCompressEnd

This function tells a compressor to end compression.

ICCompressGetFormat

This function determines the output format of a compressor.

ICCompressGetFormatSize

This function obtains the size of the output format data.

ICCompressGetSize

This function obtains the size of the compressed data.

ICCompressQuery

This function determines if a compressor can compress a specific format.

Applications use the following functions for decompressing video data:

ICDecompress

The function decompresses a single video frame.

ICDecompressBegin

This functions prepares a decompressor for decompressing data.

ICDecompressEnd

This function tells a decompressor to end decompression.

ICDecompressGetFormat

This function determines the output format of a decompressor.

ICDecompressGetFormatSize

This function obtains the size (in bytes) of the output format data.

ICDecompressGetPalette

This function obtains the palette for the output format of a decompression.

ICDecompressQuery

This function determines if a decompressor can decompress data with a specific format.

ICDecompressQuery

This function determines if a decompressor can render a specific format.

Applications use the following functions to control video decompressors that draw directly to the display:

ICDraw

This function decompresses an image for drawing.

ICDrawBegin

This function is used to start decompressing data directly to the screen.

ICDrawEnd

This function tells a decompressor to end drawing.

ICDrawFlush

This function flushes the image buffers used for drawing.

ICDrawGetTime

This function obtains the current value of the internal clock if the decompressor is handling the timing of drawing frames.

ICDrawRealize

This function tells decompressor to realize its palette used while drawing.

ICDrawSetTime

This function sets the value of the internal clock if the decompressor is handling the timing of drawing frames.

ICDrawStart

This function tells a decompressor to start its internal clock for the timing of drawing frames.

ICDrawStop

This function tells a decompressor to stop its internal clock used for the timing of drawing frames.

ICDrawWindow

This function tells a decompressor to redraw the window when it has moved.

ICGetBuffersWanted

This function obtains information about the pre-buffering needed by a compressor.

Applications use the following functions to obtain information about a compressor or decompressor and display its dialog boxes:

ICQueryAbout

This function determines if a compressor supports an about dialog box.

ICAbout

This function instructs a compressor to display its about dialog box.

ICQueryConfigure

This functions determines if a compressor supports a configuration dialog box.

ICConfigure

This function displays the configuration dialog box of the specified compressor.

ICGetInfo

This function asks a compressor for information about itself.

ICInfo

This function returns information about specific installed compressors, or it enumerates the compressors installed.

ICGetDefaultKeyFrameRate

This function obtains the default key frame rate value.

ICGetDisplayFormat

Given an input format and optionally an open compressor handle, finds the "best" format it can for displaying on the screen.

Applications use the following functions to set and retrieve the state information of a compressor or decompressor:

ICGetState

This function gets the state of a compressor.

ICGetStateSize

This function gets the size of the state data used by a compressor.

ICSetState

This function sets the state of a compressor.

Applications use the following functions to locate, open, and close a compressor or

decompressor:

ICOpen

This function opens a compressor or decompressor.

ICClose

This function closes a compressor or decompressor.

ICLocate

This function finds a compressor with specific attributes.

Applications use the following functions to install and remove a compressor or decompressor and send messages directly to it:

ICInstall

This function installs a new compressor.

ICRemove

This function removes a compressor function installed **ICInstalled**.

ICSendMessage

This function sends a message to a compressor.

Video Compression and Decompression Functions

This section contains an alphabetical list of the functions applications can use for compressing and decompressing video data. The functions are identified with the prefix IC.

ICAbout

Syntax

LRESULT ICAbout(*hic*, *hwnd*)

This function instructs a compressor or decompressor to display its about dialog box.

Parameters

HIC *hic*

Specifies the handle to the installable compressor.

HWND *hwnd*

Specifies a handle to the parent window.

Return Value Returns ICERR_OK after the compressor or decompressor displays the about dialog box. It returns ICERR_UNSUPPORTED if it does not support an about dialog box.

See Also ICQueryAbout

ICClose

Syntax **LRESULT ICClose**(*hic*)

This function closes a compressor or decompressor.

Parameters HIC *hic*

Specifies a handle to a compressor or decompressor.

Return Value Returns ICERR_OK if successful, otherwise it returns an error number.

See Also ICLocate ICOpen

ICCompress

Syntax **LRESULT ICCompress**(*hic*, *dwFlags*, *lpbiOutput*, *lpData*, *lpbiInput*, *lpBits*, *lpckid*, *lpdwFlags*, *lFrameNum*, *dwFrameSize*, *dwQuality*, *lpbiPrev*, *lpPrev*)

This function compresses a single video image.

Parameters HIC *hic*

Specifies the handle of the compressor to use.

DWORD *dwFlags*

Specifies applicable flags for the compression. The following flag is defined:

ICM_COMPRESS_KEYFRAME
Indicates that the compressor should make this frame a key frame.

LPBITMAPINFOHEADER *lpbiOutput*

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

LPVOID *lpData*

Specifies a far pointer to output data buffer.

LPBITMAPINFOHEADER *lpbiInput*

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

LPVOID *lpBits*

Specifies a far pointer to the input data buffer.

LPDWORD *lpckid*

Specifies a far pointer to a **DWORD** used to hold a chunk ID for data in the AVI file.

LPDWORD *lpdwFlags*

Specifies a far pointer to a **DWORD** holding the return flags used in the AVI index. The following flag is defined:

AVIIF_KEYFRAME
Indicates this frame is a key-frame.

LONG *lFrameNum*

Specifies the frame number.

DWORD *dwFrameSize*

Specifies the requested frame size in bytes. If set to zero, the compressor chooses the frame size.

DWORD *dwQuality*

Specifies the requested quality value for the frame.

LPBITMAPINFOHEADER *lpbiPrev*

Specifies a far pointer to a **BITMAPINFO** structure holding the previous frame's format.

LPVOID *lpPrev*

Specifies a far pointer to the previous frame's data buffer.

Return Value

This function returns ICERR_OK if successful. Otherwise, it returns an error code.

Comments

The *lpData* buffer should be large enough to hold a compressed frame. You can obtain the size of this buffer by calling **ICCompressGetSize**.

Set the *dwFrameSize* parameter to a requested frame size only if the compressor returns the VIDCF_CRUNCH flag in response to **ICGetInfo**. Without this flag, set this parameter to zero.

Set the *dwQuality* parameter to a quality value only if the compressor returns the VIDCF_QUALITY flag in response to **ICGetInfo**. Without this flag, set this parameter to

zero.

See Also ICCompressBegin, ICCompressEnd, ICCompressGetSize, ICGetInfo

---

ICCompressBegin

Syntax **LRESULT ICCompressBegin**(*hic*, *lpbiInput*, *lpbiOutput*)

This function prepares a compressor for compressing data.

Parameters HIC *hic*

Specifies a handle to a compressor.

LPBITMAPINFOHEADER *lpbiInput*

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

LPBITMAPINFOHEADER *lpbiOutput*

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

Return Value Returns ICERR_OK if the specified compression is supported, otherwise it returns ICERR_BADFORMAT if either the input or output format is not supported.

See Also ICCompress, ICCompressEnd, ICDecompressBegin, ICDrawBegin

---

ICCompressEnd

Syntax **LRESULT ICCompressEnd**(*hic*)

This function ends compression by a compressor.

Parameters HIC *hic*

Specifies a handle to the compressor.

Return Value Returns ICERR_OK if successful, otherwise it returns an error number.

See Also ICCompressBegin, ICCompress, ICDecompressEnd, ICDrawEnd

---

ICCompressGetFormat

Syntax **LRESULT ICCompressGetFormat**(*hic*, *lpbiInput*, *lpbiOutput*)

This function determines the output format of a compressor.

Parameters HIC *hic*

The compressor handle.

LPBITMAPINFOHEADER *lpbiInput*

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

LPBITMAPINFOHEADER *lpbiOutput*

Specifies a far pointer to a **BITMAPINFO** structure used to return the output format.

Return Value Returns the size of the output format.

See Also ICCompressGetFormatSize

ICCompressGetFormatSize

Syntax **LRESULT ICCompressGetFormatSize**(*hic, lpbi*)

This function obtains the size of the output format data.

Parameters HIC *hic*

Specifies a handle to a compressor.

LPBITMAPINFOHEADER *lpbi*

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

Return Value Returns the size of the output data format structure.

Comments Use this function to determine the size of the output format buffer you need to allocate when using **ICCompressGetFormat**.

See Also ICCompressGetFormat

ICCompressGetSize

Syntax **LRESULT ICCompressGetSize**(*hic, lpbiInput, lpbiOutput*)

This function obtains the size of the compressed data.

Parameters HIC *hic*

Specifies a handle to a compressor.

LPBITMAPINFOHEADER *lpbiInput*

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

LPBITMAPINFOHEADER *lpbiOutput*

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

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

See Also ICCompressQuery, ICCompressGetFormat

ICCompressQuery

Syntax **LRESULT ICCompressQuery**(*hic, lpbiInput, lpbiOutput*)

This function determines if a compressor can compress a specific format.

Parameters HIC *hic*

Specifies the compressor handle.

LPBITMAPINFOHEADER *lpbiInput*

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

LPBITMAPINFOHEADER *lpbiOutput*

Specifies a far pointer to a **BITMAPINFO** structure indicating the format of the data output. If NULL, then any output format is acceptable.

Return Value Returns ICERR_OK if the compression is supported, otherwise it returns ICERR_BADFORMAT.

See Also ICCompressGetFormat

ICConfigure

Syntax **LRESULT ICConfigure**(*hic, hwnd*)

This function displays the configuration dialog box of a compressor.

Parameters HIC *hic*

Specifies a handle to the compressor.

HWND *hwnd*

Specifies a handle to the parent window.

Return Value Returns ICERR_OK after the configuration dialog box is displayed. It returns ICERR_UNSUPPORTED if the compressor does not support a a configuration dialog box.

See Also ICQueryConfigure

ICDecompress

Syntax **LRESULT ICDecompress**(*hic*, *dwFlags*, *lpbiFormat*, *lpData*, *lpbi*, *lpBits*)

The function decompresses a single video frame.

Parameters HIC *hic*

Specifies a handle to the decompressor to use.

DWORD *dwFlags*

Specifies any applicable flags for decompression. The following flag is defined:

ICDECOMPRESS_HURRYUP
Indicates the decompressor should try to decompress at a faster rate. When an application uses this flag, it should not draw the decompressed data.

LPBITMAPINFOHEADER *lpbiFormat*

Specifies a far pointer to a **BITMAPINFO** structure containing the format of the compressed data.

LPVOID *lpData*

Specifies a far pointer to the input data.

LPBITMAPINFOHEADER *lpbi*

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

LPVOID *lpBits*

Specifies a far pointer to a data buffer for the decompressed data.

Return Value Returns ICERR_OK on success, otherwise it returns an error code.

Comments The *lpBits* parameter should point to a buffer large enough to hold the decompressed data. Applications can obtain the size of this buffer with **ICDecompressGetSize**.

See Also ICDecompressBegin, ICDecompressEnd, ICDecompressGetSize

ICDecompressBegin

Syntax **LRESULT ICDecompressBegin**(*hic*, *lpbiInput*, *lpbiOutput*)

This function prepares a decompressor for decompressing data.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPBITMAPINFOHEADER *lpbiInput*

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

LPBITMAPINFOHEADER *lpbiOutput*

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

Return Value Returns ICERR_OK if the specified decompression is supported, otherwise it returns ICERR_BADFORMAT if either the input or output format is not supported.

See Also ICDecompress, ICDecompressEnd, ICCompressBegin, ICDrawBegin

ICDecompressEnd

Syntax **LRESULT ICDecompressEnd**(*hic*)

This function tells a decompressor to end decompression.

Parameters HIC *hic*

Specifies a handle to a decompressor.

Return Value Returns ICERR_OK if successful, otherwise it returns an error number.

See Also ICDecompressBegin, ICDecompress, ICCompressEnd, ICDrawEnd

ICDecompressGetFormat

Syntax **LRESULT ICDecompressGetFormat**(*hic*, *lpbiInput*, *lpbiOutput*)

This function determines the output format of a decompressor.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPBITMAPINFOHEADER *lpbiInput*

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

LPBITMAPINFOHEADER *lpbiOutput*

Specifies a far pointer to a **BITMAPINFO** structure used to return the format of the output data.

Return Value Returns the size (in bytes) of the output format.

See Also ICDecompressGetFormatSize

ICDecompressGetFormatSize

Syntax **LRESULT ICDecompressGetFormatSize**(*hic, lpbi*)

This function obtains the size (in bytes) of the output format data.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPBITMAPINFOHEADER *lpbi*

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

Return Value Returns the size of the output data format structure.

Comments Use this function before **ICDecompressGetFormat** to find the size needed to allocate the output format buffer.

See Also ICDecompressGetFormat

ICDecompressGetPalette

Syntax **LRESULT ICDecompressGetPalette**(*hic, lpbiInput, lpbiOutput*)

This function obtains the palette for the output format of a decompression.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPBITMAPINFOHEADER *lpbiInput*

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

LPBITMAPINFOHEADER *lpbiOutput*

Specifies a far pointer to a **BITMAPINFO** structure used to return the color table. The space reserved for the color table must be at least 256 bytes.

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

See Also ICDecompressGetFormat

ICDecompressQuery

Syntax **LRESULT ICDecompressQuery**(*hic, lpbiInput, lpbiOutput*)

This function determines if a decompressor can decompress data with a specific format.

Parameters HIC *hic*

Specifies a handle to to a decompressor.

LPBITMAPINFOHEADER *lpbiInput*

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

LPBITMAPINFOHEADER *lpbiOutput*

Specifies a far pointer to a **BITMAPINFO** structure indicating the format of the output data. If NULL, any output format is acceptable.

Return Value Returns ICERR_OK if the decompression is supported, otherwise it returns ICERR_BADFORMAT.

See Also ICDecompressGetFormat

ICDecompressQuery

Syntax **LRESULT ICDecompressQuery**(*hic, lpbiInput*)

This function determines if a decompressor can render a specific format.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPBITMAPINFOHEADER *lpbiInput*

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

Return Value Returns ICERR_OK if the decompression is supported, otherwise it returns ICERR_BADFORMAT.

See Also ICDecompressQuery

ICDraw

Syntax **LRESULT ICDraw**(*hic, dwFlags, lpFormat, lpData, cbData, lTime*)

This function decompress an image for drawing.

Parameters

HIC *hic*

Specifies a handle to a decompressor.

DWORD *dwFlags*

Specifies any flags for the decompression. The following flags are defined:

ICDRAW_HURRYUP
Indicates the decompressor should try to increase its decompression rate.

ICDRAW_UPDATE
Tells the decompressor to update the screen based on data previously received. Set *lpData* to NULL when this flag is used.

LPVOID *lpFormat*

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

LPVOID *lpData*

Specifies a far pointer to the actual input data.

DWORD *cbData*

Specifies the size of the input data (in bytes).

LONG *lTime*

Specifies the time to draw this frame based on the time scale sent with **ICDrawBegin**.

Return Value

Returns ICERR_OK on success, otherwise an appropriate error number.

Comments

This function is used to decompress the image data for drawing by the decompressor. Actual drawing of frames does not occur until **ICDrawStart** is called. The application should be sure to pre-buffer the required number of frames before drawing is started (you can obtain this value with **ICGetBuffersRequired**).

See Also

ICDrawBegin, ICDrawEnd, ICDrawStart, ICDrawStop, ICGetBuffersRequired

ICDrawBegin

Syntax

LRESULT ICDrawBegin(*hic*, *dwFlags*, *hpal*, *hwnd*, *hdc*, *xDst*, *yDst*, *dxDst*, *dyDst*, *lpbi*, *xSrc*, *ySrc*, *dxSrc*, *dySrc*, *dwRate*, *dwScale*)

This function starts decompressing data directly to the screen.

Parameters

HIC *hic*

Specifies a handle to the decompressor to use.

DWORD *dwFlags*

Specifies flags for the decompression. The following flags are defined:

ICDRAW_QUERY
Determines if the decompressor can handle the decompression. The decompressor does not actually decompress the data.

ICDRAW_FULLSCREEN
Tells the decompressor to draw the decompressed data on the full screen.

ICDRAW_HDC
Indicates the decompressor should use the window handle specified by *hwnd* and the display context handle specified by *hdc* for drawing the decompressed data.

HPALETTE *hpal*

Specifies a handle to the palette used for drawing.

HWND *hwnd*

Specifies a handle for the window used for drawing.

HDC *hdc*

Specifies the display context used for drawing.

int *xDst*

Specifies the x-position of the upper-right corner of the destination rectangle.

int *yDst*

Specifies the y-position of the upper-right corner of the destination rectangle.

int *dxDst*

Specifies the width of the destination rectangle.

int *dyDst*

Specifies the height of the destination rectangle.

LPBITMAPINFOHEADER *lpbi*

Specifies a far pointer to a **BITMAPINFO** structure containing the format of the input data to be decompressed.

int *xSrc*

Specifies the x-position of the upper-right corner of the source rectangle.

int *ySrc*

Specifies the y-position of the upper-right corner of the source rectangle.

int *dxSrc*

Specifies the width of the source rectangle.

int *dySrc*

Specifies the height of the source rectangle.

DWORD *dwRate*

Specifies the data rate. The data rate in frames per second equals *dwRate* divided by *dwScale*.

DWORD *dwScale*

Specifies the data rate.

Return Value Returns ICERR_OK if it can handle the decompression, otherwise it returns ICERR_UNSUPPORTED.

Comments Decompressors use the *hwnd* and *hdc* parameters only if an application sets ICDRAW_HDC flag in *dwFlags*. It will ignore these parameters if an application sets the ICDRAW_FULLSCREEN flag. When an application uses the ICDRAW_FULLSCREEN flag, it should set *hwnd* and *hdc* to NULL.

The destination rectangle is specified only if ICDRAW_HDC is used. If an application sets the ICDRAW_FULLSCREEN flag, the destination rectangle is ignored and its parameters can be set to zero.

The source rectangle is relative to the full video frame. The portion of the video frame specified by the source rectangle will be stretched to fit in the destination rectangle.

See Also ICDraw, ICDrawEnd

ICDrawEnd

Syntax **LRESULT ICDrawEnd**(*hic*)

This function tells a decompressor to end drawing.

Parameters HIC *hic*

Specifies a handle to a compressor.

Return Value Returns ICERR_OK if successful, otherwise it returns an error number.

See Also ICDrawBegin, ICDraw, ICCompressEnd, ICDecompressEnd

ICDrawFlush

Syntax **LRESULT ICDrawFlush**(*hic*)

Flush the image buffers used for drawing.

Parameters HIC *hic*

The compressor handle.

Return Value Returns ICERR_OK on success.

See Also ICDraw, ICDrawBegin, ICDrawEnd, ICDrawStart, ICDrawStop

ICDrawGetTime

Syntax **LRESULT ICDrawGetTime**(*hic*, *lplTime*)

This function obtains the current value of the internal clock if the decompressor is handling the timing of drawing frames.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPLONG *lplTime*

Specifies a far pointer to a LONG buffer used to return the current time value. The value will be in samples (frames for video).

Return Value Returns ICERR_OK if successful.

See Also ICDrawStart, ICDrawStop, ICDrawSetTime

ICDrawRealize

Syntax **LRESULT ICDrawRealize**(*hic*, *hdc*, *fBackground*)

This function tells a decompressor to realize its palette used while drawing.

Parameters HIC *hic*

Specifies a handle to a decompressor.

HDC *hdc*

Specifies the display context used to realize the palette.

BOOL *fBackground*

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

Return Value Returns ICERR_OK if palette is realized. The compressor returns ICERR_UNSUPPORTED if it doesn't support this function.

See Also ICDrawBegin

ICDrawSetTime

Syntax **LRESULT ICDrawSetTime**(*hic*, *lpTime*)

This function sets the value of the internal clock if the installable decompressor is handling the timing of drawing frames.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPLONG *lpTime*

Specifies the current time that the compressor should be rendering. This value should be in samples. For video, this corresponds to frames.

Return Value Returns ICERR_OK if successful.

See Also ICDrawStart, ICDrawStop, ICDrawGetTime

ICDrawStart

Syntax **void ICDrawStart**(*hic*)

This function tells a decompressor to start its internal clock for the timing of drawing frames.

Parameters HIC *hic*

Specifies a handle to a compressor.

Comments This function should only be used with hardware decompressors that do their own asynchronous decompression, timing and drawing.

See Also ICDraw, ICDrawStop, ICDrawBegin, ICDrawEnd

ICDrawStop

Syntax **void ICDrawStop**(*hic*)

This function tells a decompressor to stop its internal clock used for the timing of drawing frames.

Parameters HIC *hic*

Specifies a handle to a decompressor.

Comments This function should only be used with hardware decompressors that do their own asynchronous decompression, timing and drawing.

See Also ICDraw, ICDrawStart, ICDrawBegin, ICDrawEnd

ICDrawWindow

Syntax **LRESULT ICDrawWindow**(*hic*, *prc*)

This function has a decompressor redraw the window when is has moved.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPRECT *prc*

Specifies a pointer to the destination rectangle. The destination rectangle is specified in screen coordinates.

Return Value Returns ICERR_OK if successful.

Comments This function is only supported by hardware which does its own asynchronous decompression, timing and drawing. The rectangle is set to empty if the window is totally hidden by other windows.

ICGetBuffersWanted

Syntax **LRESULT ICGetBuffersWanted**(*hic*, *lpdwBuffers*)

This function obtains information about the pre-buffering needed by a decompressor.

Parameters HIC *hic*

Specifies a handle to a decompressor.

LPDWORD *lpdwBuffers*

Specifies a far pointer to a DWORD used to return the number of samples the decompressor needs to get in advance of when they will be rendered.

Return Value Returns ICERR_OK if successful, otherwise it returns ICERR_UNSUPPORTED.

Comments This function is used only with a decompressor that uses hardware to render data and wants to ensure that hardware pipelines remain full.

ICGetDefaultKeyFrameRate

Syntax **LRESULT ICGetDefaultKeyFrameRate**(*hic*)

This function obtains the default key frame rate value.

Parameters HIC *hic*

Specifies a handle to a compressor.

Return Value Returns the default key frame rate.

ICGetDisplayFormat

Syntax **HIC ICGetDisplayFormat**(*hic*, *lpbiIn*, *lpbiOut*, *BitDepth*, *dx*, *dy*)

This function returns the "best" format available for display a compressed image. The function will also open a compressor if a handle to an open compressor is not specified.

Parameters HIC *hic*

Specifies the decompressor that should be used. If this is NULL, an appropriate compressor will be opened and returned.

LPBITMAPINFOHEADER *lpbiIn*

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

LPBITMAPINFOHEADER *lpbiOut*

Specifies a pointer to a buffer used to return the decompressed format. The buffer should be large enough for a **BITMAPINFOHEADER** and 256 color entries.

int *BitDepth*

If non-zero, specifies the preferred bit depth.

int *dx*

If non-zero, specifies the width to which the image is to be stretched.

int *dy*

If non-zero, specifies the height to which the image is to be stretched.

Return Value Returns a handle to a decompressor if successful, otherwise, it returns zero.

ICGetInfo

Syntax **LRESULT ICGetInfo**(*hic*, *lpicinfo*, *cb*)

This function obtains information about a compressor or decompressor.

Parameters HIC *hic*

Specifies a handle to a compressor or decompressor.

ICINFO FAR * *lpicinfo*

Specifies a far pointer to **ICINFO** structure used to return information about the compressor or decompressor.

DWORD *cb*

Specifies the size of the structure pointed to by *lpicinfo*.

Return Value Returns zero if successful.

ICGetState

Syntax **void ICGetState**(*hic*, *pv*, *cb*)

This function gets the state of a compressor or decompressor.

Parameters HIC *hic*

Specifies a handle to the compressor or decompressor.

LPVOID *pv*

Specifies a pointer to a buffer used to return the state data.

DWORD *cb*

Specifies the byte count for state buffer.

Comments Use **ICGetStateSize** before calling **ICGetState** to determine the size of buffer to allocate for the call.

See Also ICGetStateSize, ICSetState

ICGetStateSize

Syntax **LRESULT ICGetStateSize**(*hic*)

This function gets the size of the state data used by a compressor or decompressor.

Parameters HIC *hic*

Specifies a handle to the compressor or decompressor.

Return Value Returns the number of byte used by the state data.

Comments Use this function to get the size of the state data for the **ICGetState** and **ICSetState** buffers.

See Also ICGetState, ICSetState

ICInfo

Syntax **BOOL ICInfo**(*fccType*, *fccHandler*, *lpicinfo*)

This function returns information about specific installed compressors and decompressors, or it enumerates the compressors installed.

Parameters DWORD *fccType*

Specifies a four-character code indicating the type of compressor or decompressor.

DWORD *fccHandler*

Specifies a four-character code identifying a specific compressor or decompressor, or a number between 0 and the number of installed compressors of the type specified by *fccType*.

ICINFO FAR * *lpicinfo*

Specifies a far pointer to a **ICINFO** structure used to return information about the compressor.

Return Value Returns a compressor or decompressor handle if successful, otherwise, it returns zero.

ICInstall

Syntax **BOOL ICInstall**(*fccType*, *fccHandler*, *lParam*, *szDesc*, *wFlags*)

This function installs a new compressor.

Parameters DWORD *fccType*

Specifies a four-character code indicating the type of data used by the compressor. Use 'vidc' for video compressors.

DWORD *fccHandler*

Specifies a four-character code identifying a specific compressor.

LPARAM *lParam*

Identifies what to install. The meaning of this parameter is defined by the flags set for *wFlags*.

LPSTR *szDesc*

Specifies a pointer to a null-terminated string describing the installed compressor.

UINT *wFlags*

Specifies flags defining the contents of *lParam*. The following flags are defined:

ICINSTALL_DRIVER
Indicates *lParam* is a pointer to a null-terminated string containing the name of the compressor to install.

ICINSTALL_FUNCTION
Indicates *lParam* is a far pointer to an installable compressor function. This function should be structured like the **DriverProc** entry point function used by compressors and decompressors.

Return Value Returns a handle to a compressor or decompressor.

See Also ICRemove

ICLocate

Syntax **HIC ICLocate**(*fccType, fccHandler, lpbiIn, lpbiOut, wFlags*)

This function finds a compressor or decompressor that can handle images with the formats specified, or it finds a decompressor that can decompress an image with a specified format directly to hardware.

Parameters

DWORD *fccType*

Specifies the type of compressor or decompressor the application wants to open. For video, this is ICTYPE_VIDEO.

DWORD *fccHandler*

Specifies a single preferred handler of the given type that should be tried first. Typically, this comes from the stream header in an AVI file.

LPBITMAPINFOHEADER *lpbiIn*

Specifies a pointer to **BITMAPINFOHEADER** structure defining the input format. A compressor handle will not be returned unless it can handle this format.

LPBITMAPINFOHEADER *lpbiOut*

Specifies zero or a pointer to **BITMAPINFOHEADER** structure defining an optional decompressed format. If *lpbiOut* is nonzero, a compressor handle will not be returned unless it can create this output format.

WORD *wFlags*

Specifies any flags defining the use of the compressor or decompressor. This parameter must contain one of the following values:

ICMODE_COMPRESS
Indicates the compressor should be able to compress an image with a format defined by *lpbiIn* to the format defined by *lpbiOut*.

ICMODE_DECOMPRESS
Indicates the decompressor should be able to decompress an image with a format defined by *lpbiIn* to the format defined by *lpbiOut*.
ICMODE_DRAW
Indicates the decompressor should be able to decompress an image with a format defined by *lpbiIn* and draw it directly to hardware.

Return Value Returns a handle to a compressor or decompressor if successful, otherwise it returns zero.

ICOpen

Syntax **HIC ICOpen**(*fccType, fccHandler, wMode*)

This function opens a compressor or decompressor.

Parameters DWORD *fccType*

Specifies the type of compressor or decompressor the application wants to open. For video, this is ICTYPE_VIDEO.

DWORD *fccHandler*

Specifies a single preferred handler of the given type that should be tried first. Typically, this comes from the stream header in an AVI file.

UINT *wMode*

Specifies any flags defining the use of the compressor or decompressor. This parameter can contain the following values:

ICMODE_COMPRESS
Advises a compressor it is opened for compression.
ICMODE_DECOMPRESS
Advises a decompressor it is opened for decompression.
ICMODE_DRAW
Advises a decompressor it is opened to decompress an image and draw it directly to hardware.
ICMODE_QUERY
Advises a compressor or decompressor it is opened to obtain information.

Return Value Returns a handle to a compressor or decompressor if successful, otherwise it returns zero.

See Also ICClose ICLocate

ICQueryAbout

Syntax **BOOL ICQueryAbout**(*hic*)

This function determines if a compressor or decompressor supports an about dialog box.

Parameters HIC *hic*

Specifies the handle to the installable compressor.

Return Value Returns TRUE if the installable compressor supports an about dialog box, otherwise it returns FALSE.

See Also ICAbout

ICQueryConfigure

Syntax **BOOL ICQueryConfigure**(*hic*)

This function determines if a compressor or decompressor supports a configuration dialog box.

Parameters HIC *hic*

Specifies a handle to the compressor or decompressor.

Return Value Returns TRUE if compressor supports a configuration dialog box, otherwise it returns FALSE.

See Also ICConfigure

ICRemove

Syntax **BOOL ICRemove**(*fccType*, *fccHandler*, *wFlags*)

This function removes a compressor function installed with **ICInstall**.

Parameters DWORD *fccType*

Specifies a four-character code indicating the type of data used by the compressor. Use 'vidc' for video compressors.

DWORD *fccHandler*

Specifies a four-character code identifying a specific compressor.

UINT *wFlags*

Not used.

Return Value Returns TRUE if successful.

See Also ICInstall

ICSendMessage

Syntax **LRESULT ICSendMessage**(*hic*, *wMsg*, *dw1*, *dw2*)

This function sends a message to a compressor or decompressor.

Parameters

HIC *hic*

Specifies the handle of the compressor or decompressor to receive the message.

UINT *wMsg*

Specifies the message to send.

DWORD *dw1*

Specifies additional message-specific information.

DWORD *dw2*

Specifies additional message-specific information.

Return Value Returns a message-specific result.

ICSetState

Syntax **void ICSetState**(*hic*, *pv*, *cb*)

This function sets the state of a compressor or decompressor.

Parameters

HIC *hic*

Specifies a handle to the compressor or decompressor.

LPVOID *pv*

Specifies a pointer to the state data to set.

DWORD *cb*

Specifies the size (in bytes) of the buffer containing the state data.

See Also ICGetState, ICGetStateSize

Video Compressor and Decompressor Data Structure Reference

This section lists data structures used by video compressors and decompressors. The data structures are presented in alphabetical order. The structure definition is given, followed by a description of each field.

ICINFO

The **ICINFO** structure is filled by a video compressor 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 compressor supports quality.

VIDCF_CRUNCH
: The compressor supports crunching to a frame size.

VIDCF_TEMPORAL
: The compressor supports inter-frame compression.

VIDCF_DRAW
: The compressor supports drawing.

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

dwVersion

Specifies the version number of the compressor.

dwVersionICM

Specifies the version of the ICM supported by this compressor; 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 compressor.