

Image Compression Manager

This chapter describes the Image Compression Manager. The Image Compression Manager provides image-compression and image-decompression services to applications and other managers. If you are developing an application that works with images, you should read this chapter to familiarize yourself with the features of the Image Compression Manager. If you want to develop a compressor or decompressor for use on the Macintosh computer, see *Inside Macintosh: QuickTime Components* for information about the software interfaces that your component must support in order to work with the Image Compression Manager.

Image compression benefits you by decreasing the amount of storage required for image data, decreasing the time required to exchange image data across networks, and decreasing the time required to read data from disks and CD-ROM volumes.

This chapter is divided into the following major sections:

- “Introduction to the Image Compression Manager” contains a general introduction to the features provided by the Image Compression Manager.
- “About Image Compression” presents background information on image compression and image-compression algorithms, and it describes the features of the image compressors and decompressors supplied by Apple.
- “Using the Image Compression Manager” discusses how you can use the features of the Image Compression Manager to compress and decompress still images and image sequences—within this section are a number of shorter sections that discuss more advanced topics, including key frames, fast dithering, and compressor and decompressor components.
- “Image Compression Manager Reference” describes the data types and functions provided by the Image Compression Manager.
- “Summary of the Image Compression Manager” contains a condensed listing of the constants, data types, and functions provided by the Image Compression Manager in C and in Pascal.

Introduction to the Image Compression Manager

The Image Compression Manager provides your application with an interface for compressing and decompressing images and sequences of images that is independent of devices and algorithms.

Uncompressed image data requires a large amount of storage space. Storing a single 640-by-480 pixel image in 32-bit color can require as much as 1.2 MB. Sequences of images, like those that might be contained in a QuickTime movie, demand substantially more storage than single images. This is true even for sequences that consist of fairly small images, because the movie consists of such a large number of those images. Consequently, minimizing the storage requirements for image data is an important consideration for any application that works with images or sequences of images.

Image Compression Manager

The Image Compression Manager allows your application to

- use a common interface for all image-compression and image-decompression operations
- take advantage of any compression software or hardware that may be present in a given Macintosh configuration
- store compressed image data in pictures
- temporally compress sequences of images, further reducing the storage requirements of movies
- display compressed PICT files without the need to modify your application
- use an interface that is appropriate for your application—a high-level interface if you do not need to manipulate many compression parameters or a low-level interface that provides you greater control over the compression operation

The Image Compression Manager compresses images by invoking **image compressor components** and decompresses images using **image decompressor components**. Compressor and decompressor components are code resources that present a standard interface to the Image Compression Manager and provide image-compression and image-decompression services, respectively. The Image Compression Manager receives application requests and coordinates the actions of the appropriate components. The components perform the actual compression and decompression. Compressor and decompressor components are standard components and are managed by the Component Manager. For detailed information about creating compressor and decompressor components, see *Inside Macintosh: QuickTime Components*.

Because the Image Compression Manager is independent of specific compression algorithms and drivers, it provides a number of advantages to developers of image-compression algorithms. Specifically, compressor and decompressor components can

- present a common application interface for software-based compressors and hardware-based compressors
- provide several different compressors and compression options, allowing the Image Compression Manager or the application to choose the appropriate tool for a particular situation

Data That Is Suitable for Compression

One way to represent an image is with a pixel map, which stores a color for every pixel. For most images, however, a pixel map is an inefficient storage format. For example, a pixel map containing a solid black image would contain the color black stored over and over and over again. By compressing the image, some of this redundant information can be eliminated. The compressed image can occupy much less storage than a pixel map and can be decompressed to a pixel map when necessary.

Image Compression Manager

In addition, human perception of visual images exhibits special qualities that can be exploited to further compress image data. Image-compression algorithms take advantage of these properties to reduce the amount of information required to describe an image well enough to allow a person to see it.

A **lossless compression** technique can recreate an exact copy of the original image from the compressed form. Small changes in the image are not objectionable in most applications, however, so most compressors sacrifice some accuracy in order to further decrease the size of the compressed data. However, the compressor carefully chooses the data to omit so that the human visual system compensates for the loss and fools the user into seeing what appears to be the original image.

The Image Compression Manager works only with image data. The Image Compression Manager is primarily useful for compressing pictures that have pixel map images, such as those obtained from scanned still images or digitized video images, or from paint or three-dimensional rendering applications. You do not achieve significant compression treating pictures that are stored as groups of graphics primitives, such as those created by drawing, computer-aided design (CAD), or three-dimensional modeling applications. These applications create images in a compact format that precisely states the characteristics of the objects in the image. In fact, if you were to convert such images to pixel map representations and then compress the resulting image with the Image Compression Manager, you would probably end up with a larger, less precise image than the original. If a picture contains both primitives and pixel map image data (such as text or lines drawn over a painted or digitized image) the Image Compression Manager compresses the pixel map data and leaves the graphics primitives unchanged.

The Image Compression Manager also provides services for compressing and decompressing sequences of images or **frames** (another term for a single visual image in an image sequence). When processing a sequence, compressors may perform **temporal compression**, compressing the sequence by eliminating information that is redundant from one frame to the next. This temporal compression differs from **spatial compression**, which is performed on individual images or frames within a sequence. You may use both techniques on a single sequence.

Compressor components perform temporal compression by comparing the current frame in a sequence with the previous frame. The compressor then stores information about only those pixels that change significantly between the two images. When adjacent images contain substantially similar visual information, as is often the case in movies, temporal compression can significantly reduce the amount of data required to describe the images in the sequence. Your application indicates the desired quality level for the compressed image. The compressor uses this value to govern the extent to which it takes advantage of temporal redundancy between images. There is also a spatial quality level that you can use to control the amount of spatial compression applied to each individual image. Both of these quality values govern the amount of accuracy that is lost in the compressed image.

Image Compression Manager

Note that the Image Compression Manager does not maintain any time information for an image sequence. Rather, the Image Compression Manager maintains the order and content of the images in the sequence while the Movie Toolbox handles all timing considerations.

Storing Images

The Image Compression Manager can compress two kinds of image data: pictures and pixel maps. Pictures may be stored in memory, in a resource, or in a PICT file. Pixel maps are normally stored in a window or offscreen buffer. When compressing an image from a PICT file, the Image Compression Manager provides facilities that allow applications to spool data to and from the disk file, as appropriate to the operation. These application-provided data-loading and data-unloading functions allow arbitrarily large images to be compressed or decompressed without requiring large amounts of memory.

Applications must convert images that are not stored as pictures or pixel maps into one of these formats before compressing them. The Image Compression Manager contains several high-level functions that make it quite easy for applications to work with compressed images that are stored as PICT files. See “Working With Pictures” on page 3-28 for more information.

About Image Compression

This section provides some background information regarding image compression. This discussion has been divided into two main sections. The first, “Image-Compression Characteristics,” describes the key features you can use to choose a compression algorithm for your image data. The second, “Compressors Supplied by Apple,” discusses the compressors that are supplied with the Image Compression Manager by Apple.

Image-Compression Characteristics

There are three main characteristics by which you can judge image-compression algorithms: compression ratio, compression speed, and image quality. You can use these characteristics to determine the suitability of a given compression algorithm to your application. The following paragraphs discuss each of these attributes in more detail.

Compression Ratio

The compression ratio is equal to the size of the original image divided by the size of the compressed image. This ratio gives an indication of how much compression is achieved for a particular image.

Image Compression Manager

The compression ratio achieved usually indicates the picture quality. Generally, the higher the compression ratio, the poorer the quality of the resulting image. The trade-off between compression ratio and picture quality is an important one to consider when compressing images.

Furthermore, some compression schemes produce compression ratios that are highly dependent on the image content. This aspect of compression is called **data dependency**. Using an algorithm with a high degree of data dependency, an image of a crowd at a football game (which contains a lot of detail) may produce a very small compression ratio, whereas an image of a blue sky (which consists mostly of constant colors and intensities) may produce a very high compression ratio.

Compression Speed

Compression time and decompression time are defined as the amount of time required to compress and decompress a picture, respectively. Their value depends on the following considerations:

- the complexity of the compression algorithm
- the efficiency of the software or hardware implementation of the algorithm
- the speed of the utilized processor or auxiliary hardware

Generally, the faster that both operations can be performed, the better. Fast compression time increases the speed with which material can be created. Fast decompression time increases the speed with which the user can display and interact with images.

Image Quality

Image quality describes the fidelity with which an image-compression scheme recreates the source image data. Compression schemes can be characterized as being either lossy or lossless. Lossless schemes preserve all of the original data. **Lossy compression** does not preserve the data precisely; image data is lost, and it cannot be recovered after compression. Most lossy schemes try to compress the data as much as possible, without decreasing the image quality in a noticeable way. Some schemes may be either lossy or lossless, depending upon the quality level desired by the user.

Compressors Supplied by Apple

Apple supplies six image-compression algorithms with the Image Compression Manager. This section discusses each of these compressors and identifies their strengths and weaknesses in light of the compression characteristics just discussed. You can use this discussion as a guideline for choosing a compression algorithm for your specific situation. All the compressors support both temporal and spatial compression except for the Photo and Raw Compressors, which support only spatial compression.

The Photo Compressor

The Photo Compressor implements the **Joint Photographic Experts Group (JPEG)** algorithm for image compression. JPEG is an international standard for compressing still images. The version of JPEG supplied with QuickTime complies with the baseline International Standards Organization (ISO) standard bitstream, version 9R9.

The Photo Compressor performs best on images that vary smoothly or that do not have a large percentage of their areas devoted to edges or other types of sharp detail. This is the case for most natural (that is, nonsynthetic) images. In practice, you will find that compression ratios are highly dependent on source images, but they generally range from 5:1 to 50:1 at 24 bits per pixel, with good picture quality resulting from compression ratios between 10:1 and 20:1.

Picture quality is generally very good to excellent and is often good enough for use in demanding desktop publishing applications. Very high-resolution images obtained through the use of 24-bit color scanners would best be compressed using the Photo Compressor. This compressor is good for 8-bit grayscale images; it is not well suited to 1-bit images or non-natural images that usually have high contrast.

On a Macintosh IIsi, the Photo Compressor can compress a 24-bit, 640-by-480 pixel image at a normal quality setting in 7.5 seconds, achieving a compression ratio of 10:1. Decompressing the same image takes 6.5 seconds.

The Video Compressor

The Video Compressor employs an image-compression method developed by Apple. This method was designed to permit very fast decompression times while maintaining reasonably good picture quality. This algorithm's rapid decompression allows applications to display color images or drawings at interactive speeds. This algorithm is best suited for use with sequences of video data.

The Video Compressor is better suited to digitized video content rather than synthetically generated images. This compressor supports both spatial and temporal compression. If you use only spatial compression, you may obtain compression ratios from 5:1 to 8:1 with reasonably good quality at 24-bit pixel depths. If you use both spatial and temporal compression, the compression ratio range extends from 5:1 to 25:1.

On a Macintosh IIsi, the Video Compressor can compress a 24-bit, 640-by-480 pixel image at a normal quality setting in 3.5 seconds, achieving a compression ratio of 6.5:1. Decompressing the same image takes 1.0 second.

The Compact Video Compressor

The Compact Video Compressor is best suited to compressing 16-bit and 24-bit video sequences. It employs a lossy algorithm developed by Apple that is highly asymmetrical. In other words, it takes significantly longer to compress a frame than it does to decompress that frame. Compressing a 24-bit, 640-by-480 image on a Macintosh IIsi computer takes approximately 2.5 minutes, achieving a compression ratio of 18.5:1. Decompressing the image takes less than a second.

Compared to the Video Compressor, the Compact Video Compressor obtains higher compression ratios, better image quality, and faster playback speeds. The Compact Video Compressor can constrain data rates to user-definable levels. This is particularly important when compressing material for playback from CD-ROM discs.

For best quality results, the Compact Video Compressor should be used on raw source data that has not been compressed with a highly lossy compressor—such as the Video Compressor.

The Animation Compressor

The Animation Compressor employs a compression algorithm developed by Apple. This technique is best suited to animation and computer-generated video content. In addition, the Animation Compressor can be used to compress sequences of screen images, such as might be generated for a training application.

The Animation Compressor stores images in run-length encoded format, and it can work in either a lossy or a lossless mode. The lossless mode maintains picture content precisely, storing an animation as a series of run-length encoded images. The lossy mode loses some image quality.

The Animation Compressor's performance and achieved compression ratios are highly dependent on the type of images in a scene. The Animation Compressor is very sensitive to picture changes, and it works best on a clean image that has been generated synthetically. Images captured from videotape generally have considerable visual noise, which can corrupt the inherent similarity of the pixels and make it more difficult for the Animation Compressor to achieve good compression. This compressor works at all pixel depths.

On a Macintosh IIsi, the Animation Compressor can compress a 24-bit, 640-by-480 pixel image at a normal quality setting in 2.0 seconds, achieving a compression ratio of 1.3:1. Decompressing the same image takes 1 second.

The Graphics Compressor

The Graphics Compressor employs a compression algorithm developed by Apple. This compressor is best suited to 8-bit still images and image sequences in applications where compression ratio is more important than decompression speed.

Image Compression Manager

The Graphics Compressor is a good alternative to the Animation Compressor whenever performance is less important than compression ratio. In general, the Graphics Compressor generates a compressed image that is one-half the size of the same image compressed by the Animation Compressor. However, the Graphics Compressor can decompress the image at only half the speed of the Animation Compressor. Therefore, you should consider using the Graphics Compressor with relatively slow storage devices, such as CD-ROM discs. In these circumstances, the Graphics Compressor has sufficient time to decompress the image or image sequence.

On a Macintosh IIsi, the Graphics Compressor can compress a 640-by-480 pixel image that has been dithered to 8-bit pixel depth at a normal quality setting in 6.5 seconds, achieving a compression ratio of 2.5:1. Decompressing the same image takes 1.0 second.

The Raw Compressor

The Raw Compressor can reduce image storage requirements by converting an image from one pixel depth to another. For example, converting a 32-bit image to 16-bit format achieves a 2:1 compression ratio. The Raw Compressor can also convert a 32-bit image to 24-bit format by dropping the pad byte. This achieves a 4:3 compression with no loss of quality. The Raw Compressor accomplishes this conversion quickly, and the resulting image retains excellent image quality in most cases.

The Image Compression Manager often uses the Raw Compressor to extend the capabilities of other compressors. For example, the Photo Compressor works directly with only 32-bit color images and 8-bit grayscale images. For color images, the Image Compression Manager uses the Raw Compressor to convert the pixel depth of the original image to 32-bit color or to convert the 32-bit decompressed image to another pixel depth for display.

Image quality can deteriorate when the pixel depth is reduced; however, this technique is generally lossless when converting from a lower pixel depth to a higher depth. With 1, 2, 4, 8, and 24-bit images, the Raw Compressor allows colors to be mapped through a color table.

Note that the resulting image may be larger than the corresponding pixel image in PICT format, because QuickDraw stores PICT images in a run-length encoded format.

Note

These uncompressed QuickTime-specific PICT images cannot be used without QuickTime. ♦

Performance figures for the Raw Compressor are dependent upon the source and destination pixel depths. (The Raw Compressor is signified by the None option in the standard compression dialog box.)

Types of Images Suitable for Different Compressors

This section presents a series of graphs that indicate the amount of compression you can obtain when you compress still images with the Apple-supplied QuickTime compressors.

Note

Since some compressors make use of temporal compression, these results cannot be used to directly infer results for compressing image sequences (as in QuickTime movies). ♦

The different compressors take advantage of different properties of an image to achieve their compression; hence, the type of image being compressed significantly affects the amount of compression achieved, as well as the fidelity of the compressed image to the original.

For this comparison, three images that represent three classes of digital images are used. Figure 3-1 provides a photographic image scanned from a photographic slide. This is a natural image and contains no computer-synthesized characters or graphics elements.

Figure 3-1 24-bit photographic image



Image Compression Manager

Figure 3-2 shows a full-color image created by a three-dimensional graphics rendering program. It does not contain the detail of a natural image, but it is a full-color image that needs significantly more than 256 colors to portray it accurately. It is possible to create such an image with a full-color paint or drawing program as well as from a three-dimensional rendering program. Note also that, if an image created by these means has enough detail, it becomes more like a photographic image. Likewise, a natural image with some overlaid graphics or text may fit more closely into this category than the photographic category depending on the proportions of each type of imagery.

Figure 3-2 24-bit synthetic image



Image Compression Manager

Figure 3-3 is an example of a nondithered simple graphic image with fewer than 256 colors. The image is adequately represented by 8 bits per pixel. This image is also special in that it has large horizontal areas that are all of a single color, which is an important characteristic exploited by several compression algorithms, including the normal PICT packing used by QuickDraw.

Figure 3-3 8-bit graphic image

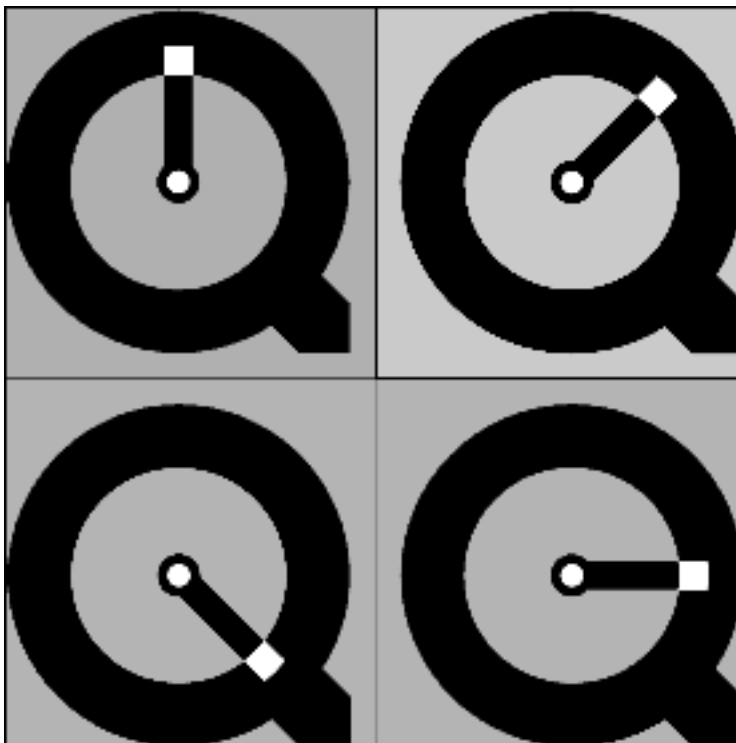


Image Compression Manager

Figure 3-4 is a natural photographic image dithered to 8 bits per pixel.

Figure 3-4 8-bit photographic image



All of the graphs show the compressed data size (in kilobytes) versus the quality of an image at minimum, low, normal, high, and maximum compression settings. The Raw Compressor is included to show the size of the image in raw pixels. The Raw Compressor is not useful for storing still images, since it does not even use the simple packing technique used by QuickDraw (notice that the 24-bit raw format is larger than the uncompressed PICT file).

Figure 3-5 provides a graph that compares compressor performance for the photographic image shown in Figure 3-1. The best compression is obtained by the Compact Video Compressor. The Photo Compressor performs as well as the Compact Video Compressor at minimum, low, and normal compression settings, but does not perform as well at high and maximum settings. However, as you might expect, the Photo Compressor retains the best image quality. The Graphics Compressor stores the image at a smaller size than the highest quality setting of the Photo Compressor, but only stores 256 colors, which significantly degrades the quality of the image. The Video Compressor does almost as well as the Photo Compressor, but the image quality is lower, because of compression artifacts and reduced color resolution. The Animation Compressor retains the color resolution and detail of the image when storing millions of colors and the detail when storing thousands of colors, but it does not achieve nearly as much compression as the other compressors.

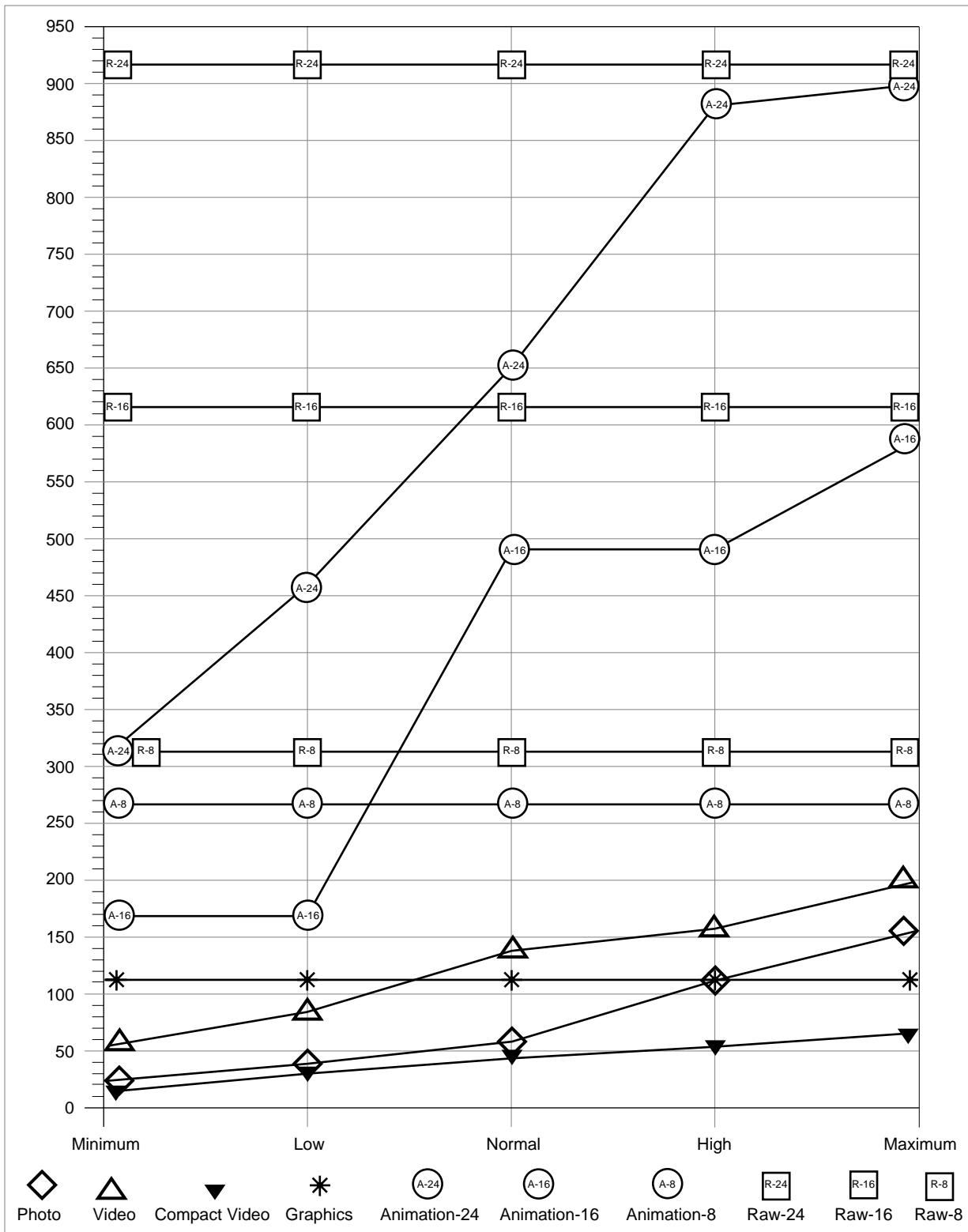
Figure 3-5 Compressor performance for a 921 KB, 24-bit, photographic image

Image Compression Manager

The graph in Figure 3-6 compares compressor performance for the full-color, computer-synthesized image shown in Figure 3-2. The Compact Video Compressor again achieves the best overall compression, followed by the Photo and Video Compressors. Again the Graphics Compressor cannot accurately represent all of the colors of the image and is not suitable for use on this type of image. With this image, the Animation Compressor does better than it did with the natural image, and it may be suitable if space constraints are not as important as speed constraints. Because computer-generated images tend to have smoother color gradations than natural images, the loss of color resolution with the Video Compressor and the 16-bit Raw and Animation Compressors is more apparent.

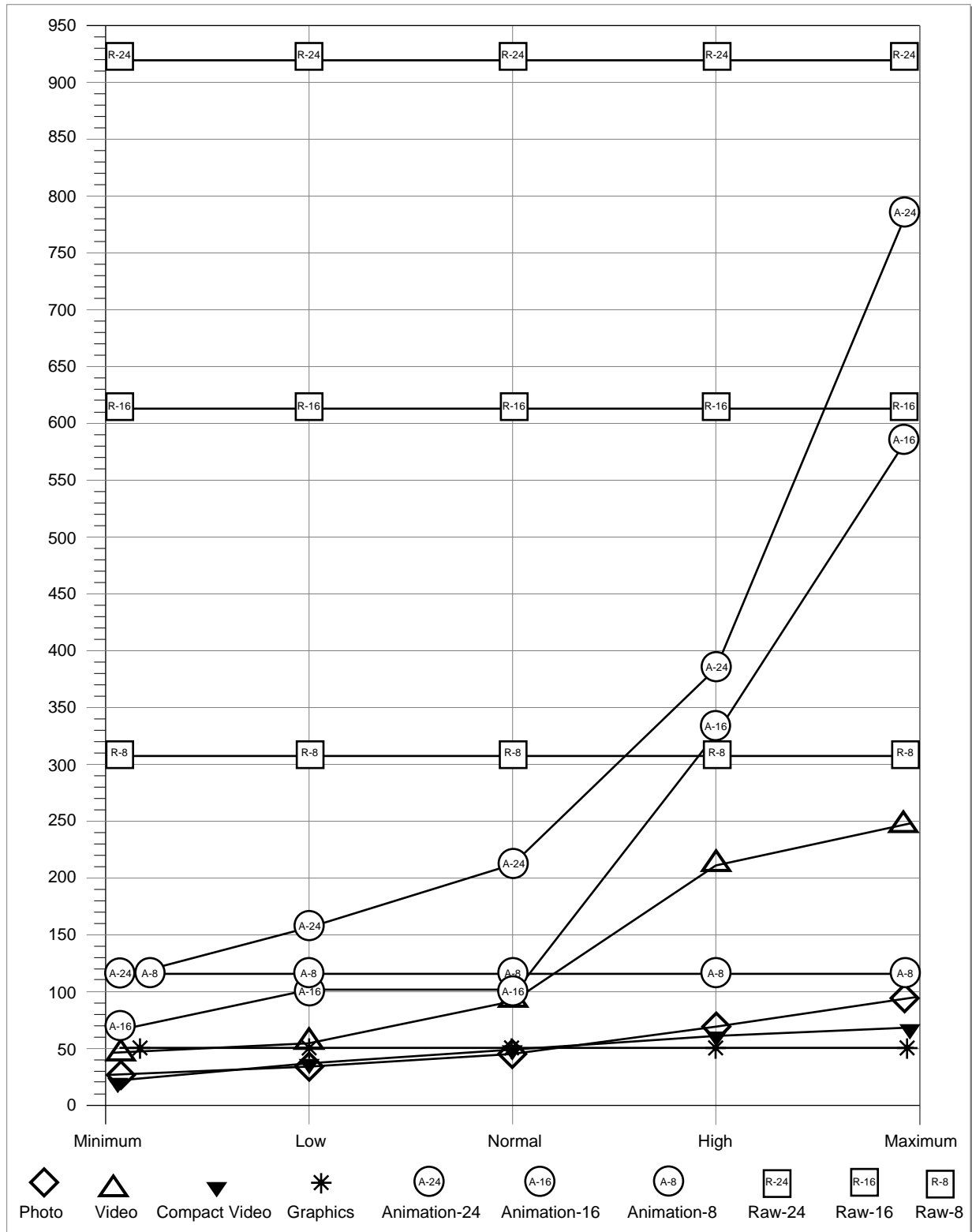
Figure 3-6 Compressor performance for a 502 KB, 24-bit, synthetic image

Image Compression Manager

Figure 3-7 compares compressor performance for the simple graphic image shown in Figure 3-3. The Graphics Compressor is the only reasonable choice. Not only does it produce the best compression, but also it stores the image without losing any of the image's detail, since there are fewer than 256 colors in the source image. The Photo and Compact Video Compressors get some compression, but do not store the image as accurately as the Graphics Compressor. The Video Compressor stores the image even less accurately and does not compress the image well at all. The Animation Compressor also does not store the image with complete accuracy at 16 or even 24 bits per pixel, and the resulting files are much larger than the uncompressed PICT. Although the 8-bit Animation Compressor does store the image accurately, it only achieves half as much compression as the Graphics Compressor and its file is also larger than the original PICT.

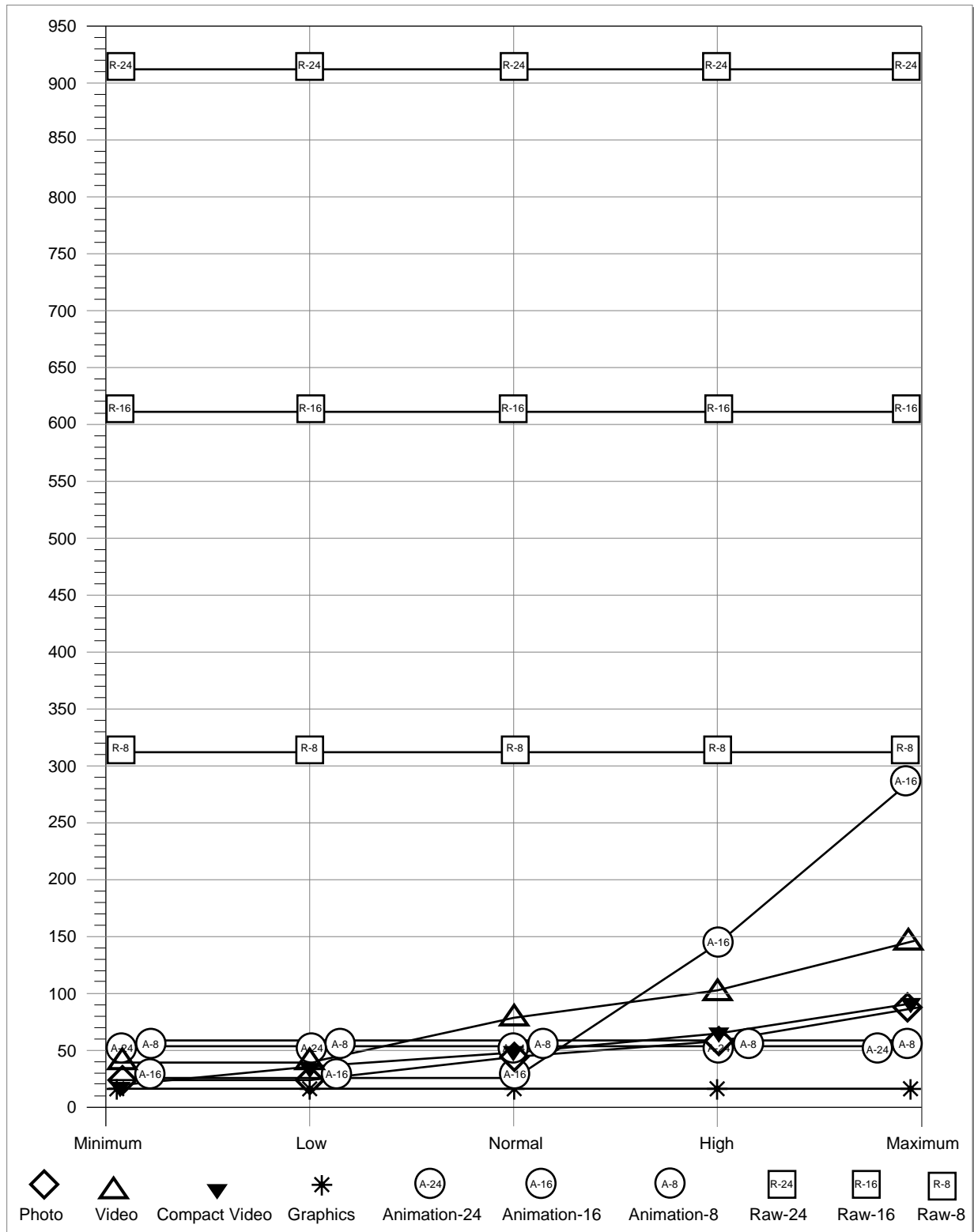
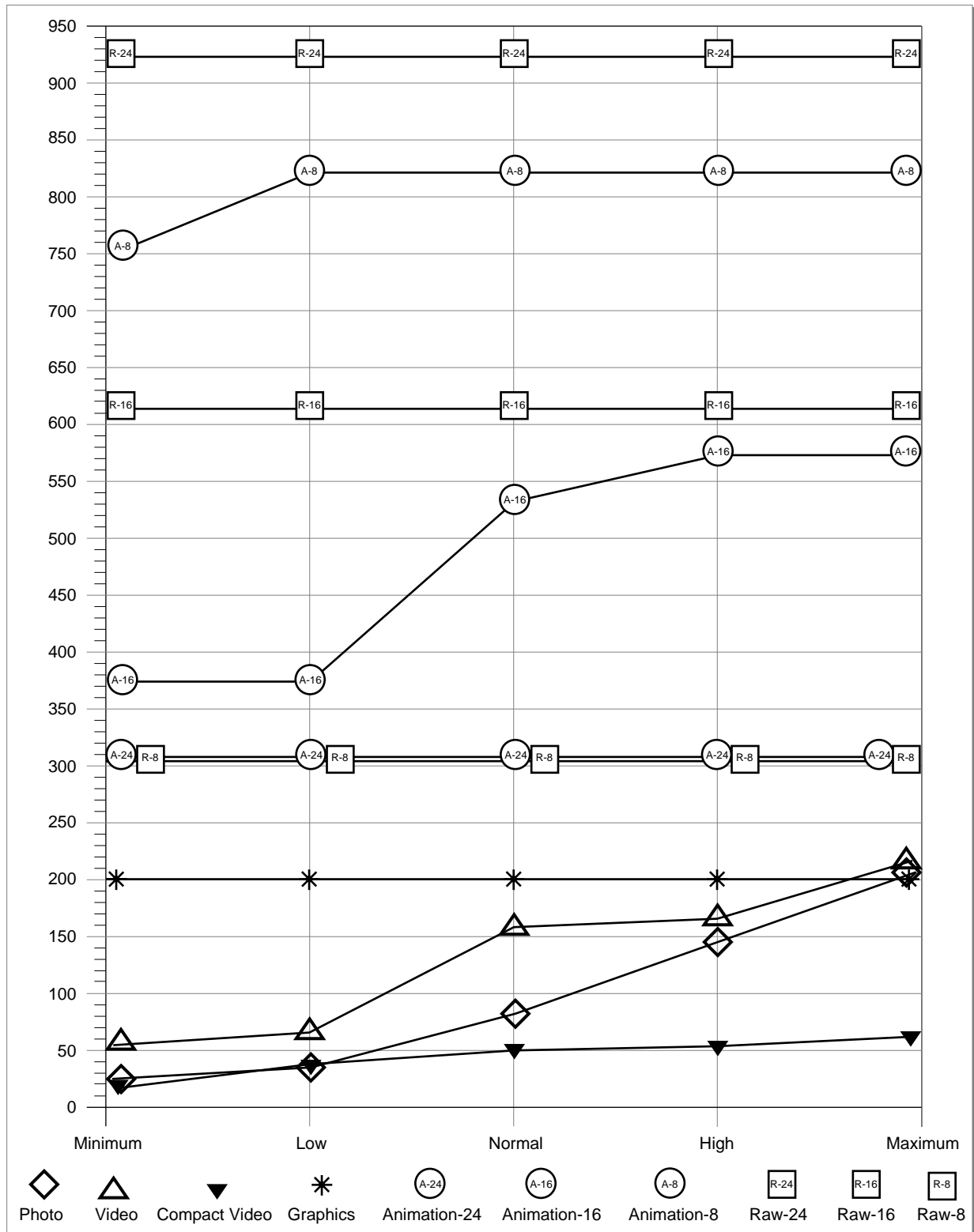
Figure 3-7 Compressor performance for a 30 KB, 8-bit, graphic image

Image Compression Manager

The graph in Figure 3-8 compares performance for the 8-bit, dithered, photographic image shown in Figure 3-4. The best results are obtained by the Compact Video Compressor. The rest of the results are almost the same as for the full-color, natural image shown in Figure 3-5, but this time the Graphics Compressor stores the image exactly, since it had only 256 colors to start with. The other compressors do almost as well as they did for the full-color, natural image, but the compression for all of them is a bit worse, due to the added artifacts introduced when the image was converted to 8 bits per pixel. The 16-bit and 24-bit versions of the Animation Compressor do not make sense for this image, since their results are always larger than the original PICT. The Photo and Video Compressors still do well on this image, but they do lose some detail that the Graphics Compressor retains. The losses are minor, however, and the sizes approach the size of the Graphics Compressor's image only at high-quality settings, where the losses are negligible.

Figure 3-8 Compressor performance for a 302 KB, 8-bit, dithered, photographic image

Using the Image Compression Manager

This section discusses several of the ways your application may use the Image Compression Manager to compress and decompress images and sequences of images.

Getting Information About Compressors and Compressed Data

Use the Gestalt environmental selector `gestaltCompressionMgr` to determine whether the Image Compression Manager is available. Gestalt returns a 32-bit value indicating the version of the Image Compression Manager that is installed. This return value is formatted in the same way as the value returned by the `CodecManagerVersion` function (described on page 3-66), and it contains the version number specified as an integer value.

```
#define gestaltCompressionMgr 'icmp'
```

The Image Compression Manager provides a number of functions that allow your application to obtain information about the facilities available for image compression or about compressed images. Your application may use some of these functions to select a specific compressor or decompressor for a given operation or to determine how much memory to allocate to receive a decompressed image. In addition, your application may use some of these functions to determine the capabilities of the components that are available on the user's computer system. You can then condition the options your program makes available to the user based on the user's system configuration. See "Getting Information About Compressor Components," which begins on page 3-66, and "Getting Information About Compressed Data," which begins on page 3-71, for detailed descriptions of these functions.

Working With Pictures

The Image Compression Manager provides a set of functions that allow applications to work easily with compressed pictures stored in version 2 PICT files. These functions constitute a high-level interface to image compression and decompression. Applications that require little control over the compression process may use these functions to display pictures that contain compressed image data.

Existing programs can display (without changes) pictures that contain compressed image data. When the Image Compression Manager is installed on a system, it installs a new `StdPix` graphics function (see page 3-141 for more information on the `StdPix` graphics function). This function handles all requests to display compressed images. Whenever an application issues the standard `QuickDraw DrawPicture` routine

Image Compression Manager

(described in *Inside Macintosh: Imaging*) to display an image that contains compressed image data, the `StdPix` function decompresses the image by invoking the Image Compression Manager. The function then delivers the decompressed image to the application.

The Image Compression Manager also provides a simple mechanism for creating a picture that contains compressed image data. For example, to place an existing compressed image into a picture, your application could open the picture with QuickDraw's `OpenPicture` (or `OpenCPicture`) function and then call the Image Compression Manager's `DecompressImage` function, as if you were going to display the image. The Image Compression Manager places the compressed image and the other data that describe the image into the picture for you.

The Image Compression Manager stores the following information about a compressed picture:

- the image description, which describes the compression format and characteristics of the compressed image data
- the compressed data for the image
- the transfer mode (source copy mode, dither copy mode, and so on)
- the matte pixel map
- the mask region
- the mapping matrix
- the source rectangle of the image

The Image Compression Manager stores this information in the picture as a new PICT opcode (described in the following paragraphs). When an application draws the compressed picture on a Macintosh computer that is running the Image Compression Manager, the `StdPix` function instructs the Image Compression Manager to decompress the image. If an application tries to read a picture file that contains compressed data on a Macintosh that does not have the Image Compression Manager installed, the system ignores the new opcodes and displays a message that indicates that the user needs QuickTime in order to display the compressed image data. The message states “QuickTime™ and a <Codec Name> decompressor are needed to see this picture”.

The Color QuickDraw version 2 picture format includes PICT opcodes for compressed and uncompressed QuickTime images. (An opcode is a hexadecimal number that represents drawing commands and the parameters that affect those drawing commands in a picture.) For more information on the version 2 picture format, see the chapter “Color QuickDraw” in *Inside Macintosh: Imaging*.

The PICT opcodes for compressed and uncompressed QuickTime images are

- opcode \$8200, which signals a compressed QuickTime image
- opcode \$8201, which signals an uncompressed QuickTime image

Table 3-1 gives an overview of the opcode for QuickTime compressed pictures.

Table 3-1 Fields of the PICT opcode for compressed QuickTime images

Field name	Description	Data size (in bytes)
Opcode	Compressed picture data	2
Size	Size in bytes of data for this opcode	4
Version	Version of this opcode	2
Matrix	3 by 3 fixed transformation matrix	36
MatteSize	Size of matte data in bytes	4
MatteRect	Rectangle for matte data	8
Mode	Transfer mode	2
SrcRect	Rectangle for source	8
Accuracy	Preferred accuracy	4
MaskSize	Size of mask region in bytes	4

▲ **WARNING**

Do not attempt to read opcodes directly. For compatibility reasons, use Toolbox routines to access this information. ▲

The MaskSize field of opcode \$8200 is followed by five variable fields:

- The matte image description, which contains the image description structure for the matte. The variable size is specified in the first long integer in the opcode. This field is not included if the MatteSize field is 0.
- The matte data, which contains the compressed data for the matte. The size of this field is defined by the MatteSize field described in Table 3-1. This field is not included if the MatteSize field is 0.
- The mask region, which contains the region for masking. The size of this variable is defined by the MaskSize field described in Table 3-1. This field is not included if the MaskSize field is 0.
- The image description structure for this data. The size of this variable is specified in the first long integer in the idSize field of this image description.
- The image data, which contains the compressed data for the image. The size of the image data is specified in the image description's dataSize field.

See “The Image Description Structure” beginning on page 3-53 for details on the idSize and dataSize fields.

Table 3-2 provides an overview of the structure of uncompressed QuickTime images.

Table 3-2 Fields of the PICT opcode for uncompressed QuickTime images

Field name	Description	Data size (in bytes)
Opcode	Uncompressed picture data	2
Size	Size in bytes of data for this opcode	4
Version	Version of this opcode	2
Matrix	3 by 3 fixed transformation matrix	36
MatteSize	Size of matte data in bytes	4
MatteRect	Rectangle for matte data	8

The `MatteRect` field of opcode \$8201 is followed by three variable fields and a subopcode:

- The matte image description, which contains the image description structure for the matte. The size of this variable is specified in the first long integer in this opcode. This field is not included if the `MatteSize` field is 0.
- The matte data, which contains information for the matte. The size of this variable is defined by the `MatteSize` field.
- A subopcode (2 bytes in length) which describes the image and mask and is entirely within the other opcode. Its size is included in the size for the main opcode; hence it is not included if the QuickTime opcode is skipped. This subopcode can be either \$98, \$99, \$9A, or \$9B.
- The data for the subopcode variable which contains information for the image.

Compressing Images

The Image Compression Manager provides a rich set of functions that allow applications to compress images. Some of these functions present a straightforward interface that is suitable for applications that need little control over the compression operation. Others permit applications to control the parameters that govern the compression operation.

This section describes the basic steps that your application follows when compressing a single frame of image data. Following this discussion, Listing 3-1 shows a sample function that compresses an image.

First, determine the parameters for the compression operation. Typically, the user specifies these parameters in a user dialog box you may supply via the standard compression dialog component. For comprehensive details, see the chapter “Standard Image-Compression Dialog Components” in *Inside Macintosh: QuickTime Components*. Your application may choose to give the user the ability to specify such parameters as the compression algorithm, image quality, and so on.

Image Compression Manager

Your application may give the user the option to specify a compression algorithm based on an important performance characteristic. For example, the user may be most concerned with size, speed, or quality. The Image Compression Manager allows your application to choose the compressor component that meets the specified criterion.

To determine the maximum size of the resulting compressed image, your application should then call the Image Compression Manager's `GetMaxCompressionSize` function (described on page 3-72). You provide the specified compression parameters to this function. In response, the Image Compression Manager invokes the appropriate compressor component to determine the maximum number of bytes required to store the compressed image. Your application should then reserve sufficient memory to accommodate the compressed image or use a data-unloading function to spool the compressed data to disk (see "Spooling Compressed Data" beginning on page 3-48 for more information about data-unloading functions).

Once the user has specified the compression parameters and your application has established an appropriate environment for the operation, call the `CompressImage` (or `FCompressImage`) function to compress the image. Use the `CompressImage` function (described on page 3-77) if your application does not need to control all the parameters governing compression. If your application needs access to other compression parameters, use the `FCompressImage` function (described on page 3-79).

The Image Compression Manager manages the compression operation and invokes the appropriate compressor. The manager returns the compressed image and its associated image description structure to your application. Note that the image description structure contains a field indicating the size of the resulting image.

Note

You should use the standard compression dialog component to set up the parameters for compression. See the chapter "Standard Image-Compression Dialog Components" in *Inside Macintosh: QuickTime Components* for details. ♦

Listing 3-1 Compressing and decompressing an image

```
#include <Types.h>
#include <Traps.h>
#include <Memory.h>
#include <Errors.h>
#include <FixMath.h>
#include "Movies.h"
#include "ImageCompression.h"
#include "StdCompression.h"

#define kMgrChoose 0
PicHandle GetQTCompressedPict (PixmapHandle myPixmap);
```


Image Compression Manager

```

PicHandle GetQTCompressedPict( PixMapHandle myPixMap )
{
    long                maxCompressedSize = 0;
    Handle              compressedDataH = nil;
    Ptr                 compressedDataP;
    ImageDescriptionHandle imageDescH = nil;
    OSErr               theErr;
    PicHandle           myPic = nil;
    Rect                bounds = (**myPixMap).bounds;
    CodecType           theCodecType = 'jpeg';
    CodecComponent       theCodec = (CodecComponent)anyCodec;
    CodecQ              spatialQuality = codecNormalQuality;
    short               depth = 0; /* let ICM choose depth */

    theErr = GetMaxCompressionSize( myPixMap, &bounds, depth,
                                    spatialQuality, theCodecType,
                                    (CompressorComponent)theCodec,
                                    &maxCompressedSize);

    if ( theErr ) return nil;

    imageDescH = (ImageDescriptionHandle)NewHandle(4);
    compressedDataH = NewHandle(maxCompressedSize);
    if ( compressedDataH != nil && imageDescH != nil )
    {
        MoveHHi(compressedDataH);
        HLock(compressedDataH);
        compressedDataP = StripAddress(*compressedDataH);

        theErr = CompressImage( myPixMap,
                                &bounds,
                                spatialQuality,
                                theCodecType,
                                imageDescH,
                                compressedDataP);

        if ( theErr == noErr )
        {
            ClipRect(&bounds);
            myPic = OpenPicture(&bounds);
            theErr = DecompressImage( compressedDataP,
                                      imageDescH,
                                      myPixMap,

```

Image Compression Manager

```

                                &bounds,
                                &bounds,
                                srcCopy,
                                nil );

        ClosePicture();
    }
    if ( theErr
        || GetHandleSize((Handle)myPic) == sizeof(Picture) )
    {
        KillPicture(myPic);
        myPic = nil;
    }
}
if (imageDescH) DisposeHandle( (Handle)imageDescH);
if (compressedDataH) DisposeHandle( compressedDataH);

return myPic;
}

```

Decompressing Images

“Working With Pictures,” which begins on page 3-28, discusses how applications can display compressed images that are stored as pictures by calling the `DrawPicture` function. The Image Compression Manager also provides functions that allow your application to display single-frame compressed images. As with image compression, your application can choose to specify all the parameters that govern the operation, or it can leave many of these choices to the Image Compression Manager.

This section describes the steps your application must follow to decompress an image into a pixel map.

First, your application determines where to display the decompressed image. Your application must specify the destination graphics port to the Image Compression Manager. In addition, you may indicate that only a portion of the source image is to be displayed. You describe the desired portion of the image by specifying a rectangle in the coordinate system of the source image. You can determine the size of the source image by examining the image description structure associated with the image (see “The Image Description Structure” on page 3-53 for more information about image description structures).

Your application may also specify that the image is to be mapped into the destination graphics port. The Image Compression Manager provides two mechanisms for mapping images during decompression. The `DecompressImage` function (described on page 3-82) accepts a second rectangle as a parameter. During decompression the Image Compression Manager maps the desired image to the destination rectangle, scaling the resulting image as appropriate to fit the destination rectangle. The `FDecompressImage` function (described on page 3-83) allows your application to specify a mapping matrix

Image Compression Manager

for the operation. Currently, the Image Compression Manager supports only scaling and translation matrix operations.

Your application can invoke further effects by specifying a mask region or blend matte for the image. Mask regions and mattes control which pixels in the source image are drawn to the destination. **Mask regions** define the part of the source image that is displayed. During decompression the Image Compression Manager displays only those pixels in the source image that correspond to bits in the mask that are set to 1. Mask regions must be defined in the destination coordinate system.

Blend mattes contain several bits per pixel and are defined in the coordinate system of the source image. Mattes provide a mechanism for mixing two images. The Image Compression Manager displays the weighted average of the source and destination based on the corresponding pixel in the matte (this feature is fully functional in System 7 and is approximated in System 6).

Decompress the image by calling the Image Compression Manager's `DecompressImage` or `FDecompressImage` function. Your application must provide an image description structure along with the other parameters governing the operation. Use the `DecompressImage` function for simple decompression operations. If your application needs greater control, use the `FDecompressImage` function. See "Working With Images" which begins on page 3-77, for detailed descriptions of these functions.

The Image Compression Manager manages the decompression operation and invokes the appropriate decompressor component. The manager returns the decompressed image to the location specified by your application.

Compressing Sequences

The Image Compression Manager also provides functions that allow your application to compress and decompress sequences of images, such as might constitute a QuickTime movie. The tools provided by the Image Compression Manager focus on image compression and decompression and on the ordering of the images in a sequence, not on timing considerations. Use the Movie Toolbox to handle all the issues relating to the amount of time each image should be shown on the screen. For information on decompressing image sequences, see the next section, "Decompressing Sequences."

A series of images can be compressed as a sequence if those images share an image description. That is, each image in the sequence must have the same compressor type, pixel depth, color lookup table, and boundary dimensions. To take best advantage of temporal compression, the images should also be related to each other (like frames in a movie), but this relationship is not necessary for them to be grouped as a sequence. If you create a sequence from completely unrelated images, you may not be able to achieve significant temporal compression.

When compressing image sequences, your application must perform several steps in addition to those required for single-frame image compression. This section describes a typical function for compressing an image sequence. Note that much of the setup processing is the same as that performed for single-frame images.

Image Compression Manager

First, determine the parameters for the compression operation. As with single-image compression, the user may specify these parameters in a dialog box you can supply via the standard image-compression dialog component (see the chapter “Standard Image-Compression Dialog Components” in *Inside Macintosh: QuickTime Components* for details). Your application may choose to give the user the ability to specify such parameters as the compression algorithm, image quality, and so on. Note that image sequences require additional parameters, such as temporal quality.

Your application may give the user the option of specifying a compression algorithm based on an important performance characteristic. For example, the user may be most concerned with size, speed, or accuracy. The Image Compression Manager allows your application to choose the compressor component that meets the specified criterion.

Your application signals its intention to compress an image sequence by issuing the Image Compression Manager’s `CompressSequenceBegin` function (see page 3-110 for more information about this function). At this time your application specifies many of the parameters that govern the sequence-compression operation. When you set the compression parameters and the `temporalQuality` parameter is not 0, then be sure to set the value of either the `codecFlagUpdatePrevious` or `codecFlagUpdatePreviousComp` flag to 1 in the `flags` parameter of the `CompressSequenceBegin` function.

Once you have started the sequence, you then compress each image in the sequence by performing the following steps:

1. Your application must call the Image Compression Manager’s `GetMaxCompressionSize` function to determine the maximum size of the compressed data that will result from the current image (see “Getting Information About Compressed Data” on page 3-71 for more information about this function). You provide the specified compression parameters to this function. In response, the Image Compression Manager invokes the appropriate compressor component to determine the number of bytes required to store the largest compressed image in the sequence. Your application should then reserve sufficient memory to accommodate that compressed image. You can use this returned value until you change the settings of the compression parameters.
2. Your application must call the `CompressSequenceFrame` function to compress the image (see “Working With Sequences” on page 3-110 for more information about this function). It may be necessary or desirable for your application to change one or more of the compression parameters while processing a sequence. The Image Compression Manager provides several functions that allow your application to modify such parameters as the spatial or temporal quality or the data-unloading function. See “Changing Sequence-Compression Parameters” on page 3-124 for more information about these functions.
3. The Image Compression Manager manages the compression operation and invokes the appropriate compressor. The manager returns the compressed image and its associated image description to your application.
4. Your application is then free to store the compressed image with the others in the sequence.

After the entire sequence is compressed, you end the process by calling the `CDSequenceEnd` function (see page 3-123 for more information about this function).

Decompressing Sequences

The Movie Toolbox handles the details of displaying compressed image sequences that are stored in QuickTime movies. (For details, see the chapter “Movie Toolbox” in this book.) However, if you want to work with sequences in your application, the Image Compression Manager provides tools for decompressing image sequences. As with still-image compression, decompressing sequences requires additional effort on the part of your application. In addition, there are some processing considerations that are particular to sequence decompression. This section describes the steps necessary to decompress an image sequence. Then it discusses several points you should consider before decompressing a sequence.

When decompressing an image sequence, your application must first determine where to display the decompressed sequence. Your application must specify the destination graphics port to the Image Compression Manager. In addition, you may indicate that only a portion of the source image is to be displayed. You describe the desired portion of the image by specifying a rectangle in the coordinate system of the source image. You can determine the size of the source image by examining the image description structure associated with the image (see “The Image Description Structure” on page 3-53 for more information about image description structures).

Your application may also specify that the image is to be mapped into the destination graphics port. The `DecompressSequenceBegin` function (described on page 3-117) allows your application to specify a mapping matrix for the operation.

Your application can invoke additional effects by specifying a mask region or blend matte for the image. Mask regions and mattes control which pixels in the source image are drawn to the destination. Mask regions must be defined in the destination coordinate system. During decompression the Image Compression Manager displays only those pixels in the source image that correspond to bits in the mask that are set to 1. Mattes contain several bits per pixel and are defined in the coordinate system of the source image. Mattes provide a mechanism for blending pixels from source images.

Your application signals its intention to decompress an image sequence by issuing the Image Compression Manager’s `DecompressSequenceBegin` function (see page 3-117 for more information about this function). At this time your application specifies many of the parameters that govern the sequence-decompression operation. The Image Compression Manager, in turn, allocates system resources that are necessary for the operation.

Once you have started the sequence, you then decompress each image in the sequence. Call the `DecompressSequenceFrame` function to decompress the image (described on page 3-120). It may be necessary or desirable for your application to change one or more of the decompression parameters while processing a sequence. The Image Compression Manager provides several functions that allow your application to modify such parameters as the accuracy, the transformation matrix, or the data-loading function. See

Image Compression Manager

“Changing Sequence-Decompression Parameters” beginning on page 3-133 for more information about these functions.

The Image Compression Manager manages the decompression operation and invokes the appropriate compressor component. The manager returns the decompressed image to the location specified by your application and applies any effects you may have specified.

After the entire sequence is decompressed, you end the process by calling the `CDSequenceEnd` function (described on page 3-123).

Decompressing Still Images From a Sequence

Your application can, of course, decompress individual images from a sequence. When doing so, you must be careful to select only those frames that do not depend on other frames. That is, do not decompress frames from a sequence that has been temporally compressed unless you first decompress all the frames in sequence starting from the preceding key frame (see “Defining Key Frame Rates” on page 3-51 for more information on key frames in image sequences). In general, you should decompress images from sequences as sequences, rather than as individual frames.

Using Screen Buffers and Image Buffers

There are two special buffers associated with decompressing an image sequence: a screen buffer and an image buffer. The Image Compression Manager uses the screen buffer to reduce tearing artifacts that result when an image cannot be decompressed to the screen quickly enough. Tearing manifests itself when your eye sees parts of consecutive images simultaneously. Screen buffers should be the same size and pixel depth as the destination. This provides the fastest screen update speed. The compressor decompresses the image to the screen buffer, performing the time-consuming tasks associated with decompression. When the image is fully decompressed, the compressor quickly copies the image to the screen. Few sequences require the use of a screen buffer. You must determine whether it is appropriate to your application.

The Image Compression Manager uses image buffers when decompressing sequences that have been temporally compressed and therefore contain key frames. Image buffers are especially useful when you want to skip to random frames within a sequence. Random frame access in temporally compressed sequences forces the compressor to decompress all the frames between the nearest preceding key frame and the desired frame. Reconstructing the frame in this manner on the screen can result in jerky sequence display. As an alternative, the compressor can reconstruct the frame in the offscreen image buffer and then copy it to the screen when appropriate. Image buffers are allocated at an appropriate depth and size for the decompressor.

Your application can control the use of the image buffer by the compressor component. For example, you can force the compressor to draw images only to the image buffer, not to the screen. In this manner you can use the image buffer to build up sequences without making the process visible. You can also control when the compressor uses the image

buffer. You may need to do this when your program is decompressing directly to the screen and suddenly is prevented from doing so (for example, when your window becomes hidden).

A Sample Program for Compressing and Decompressing a Sequence of Images

The sample program presented in this section illustrates the processes described in the previous sections. The program has been divided into several functions. Listing 3-2 shows the main program.

Listing 3-2 Compressing and decompressing a sequence of images: The main program

```
WindowPtr    displayWindow;    /* window in which to display
                                sequence */
Rect          windowRect;      /* rectangle of displayWindow */

main (void)
{
    WindowPtr    displayWindow;
    Rect          windowRect;

    InitGraf (&thePort);
    InitFonts ();
    InitWindows ();
    InitMenus ();
    TEInit ();
    InitDialogs (nil);

    SetRect (&windowRect, 0, 0, 256, 256);
    OffsetRect (&windowRect, /* middle of screen */
                ((qd.screenBits.bounds.right - qd.screenBits.bounds.left) -
                 windowRect.right) / 2,
                ((qd.screenBits.bounds.bottom - qd.screenBits.bounds.top) -
                 windowRect.bottom) / 2);
    displayWindow = NewCWindow (nil, &windowRect,
                                "\pImage", true, 0,
                                (WindowPtr)-1, true, 0);

    if (displayWindow)
    {
        SetPort (displayWindow);
        SequenceSave ();
    }
}
```

Image Compression Manager

```

        SequencePlay ();
    }
}

```

A Sample Function for Saving a Sequence of Images to a Disk File

The `SequenceSave` function shown in Listing 3-3 saves a sequence of images to a disk file. This function creates and opens a disk file for the image sequence, calls the `CompressSequence` function to create and compress the image sequence into the file, and then calls the `MakeMyResource` function to save the image description resource in the file, so that the sequence can be played back later. For details on `CompressSequence`, see the next section.

The data for each frame is written to the data fork of the disk file, preceded by a long word that contains the number of bytes of data for that frame. A description of the compressed images in the sequence is stored in a 'SEQU' resource in the same file with a resource ID of 128 or 129. This description is simply the image description structure maintained by the Image Compression Manager.

The image for each frame of the sequence is drawn into an offscreen graphics world that the `SequenceSave` function creates in the `currWorld` variable. `SequenceSave` calls the `DrawOneFrame` function (described in the next section) to draw each frame's image into the `currWorld` variable. Before any of the frames of the sequence are drawn, the Image Compression Manager is prepared to compress a sequence of images through the `CompressSequence` function.

Listing 3-3 Compressing and decompressing a sequence of images: Saving a sequence to a disk file

```

void SequenceSave (void)
{
    long                filePos;
    StandardFileReply    fileReply;
    short               dfRef = 0;
    OSErr               error;
    ImageDescriptionHandle description = nil;

    StandardPutFile ("\p", "\pSequence File", &fileReply);
    if (fileReply.sfGood)
    {
        if (! (fileReply.sfReplacing))
        {
            error = FSpCreate (&fileReply.sfFile, 'SEQM', 'SEQU',
                               fileReply.sfScript);
            CheckError (error, "\pFSpCreate");
        }
    }
}

```


Image Compression Manager

```

        error = FSpOpenDF (&fileReply.sfFile, fsWrPerm, &dfRef);
        CheckError (error, "\pFSpOpenDF");

        error = SetFPos (dfRef, fsFromStart, 0);
        CheckError (error, "\pSetFPos");

        CompressSequence (&dfRef, &description);
        error = GetFPos (dfRef, &filePos);
        CheckError (error, "\pGetFPos");

        error = SetEOF (dfRef, filePos);
        CheckError (error, "\pSetEOF");

        FSClose (dfRef);
        FlushVol (nil, fileReply.sfFile.vRefNum);

        MakeMyResource (fileReply, description);
        if (description != nil)
            DisposeHandle ((Handle) description);
    }
}

void MakeMyResource ( StandardFileReply fileReply,
                    ImageDescriptionHandle description)
{
    OSErr    error;
    short    rfRef;
    Handle    sequResource;

    FSpCreateResFile (&fileReply.sfFile, 'SEQM', 'SEQU',
                    fileReply.sfScript);

    error = ResError();
    if (error != dupFNErr)
        CheckError (error, "\pFSpCreateResFile");

    rfRef = FSpOpenResFile (&fileReply.sfFile, fsRdWrPerm);
    CheckError (ResError (), "\pFSpOpenResFile");

    SetResLoad (false);
    sequResource = Get1Resource ('SEQU', 128);
    if (sequResource)
        RmveResource (sequResource);
}

```

Image Compression Manager

```

    SetResLoad (true);
    sequResource = (Handle) description;
    error = HandToHand (&sequResource);
    CheckError (error, "\pHandToHand");

    AddResource (sequResource, 'SEQU', 128, "\p");
    CheckError (ResError (), "\pAddResource");
    UpdateResFile (rfRef);
    CheckError (ResError (), "\pUpdateResFile");

    CloseResFile (rfRef);
}

```

A Sample Function for Creating, Compressing, and Drawing a Sequence of Images

Listing 3-4 shows the `CompressSequence` function, which creates and then compresses the image sequence. `CompressSequenceBegin` informs the Image Compression Manager which compressor (of type `codectype`) to use, what the desired compression quality is, the key frame rate, the portion of the image to compress (in this example, the entire image is compressed), and the image to be compressed (in this example, the pixel map [of type `PixMap`] in the `currWorld` variable).

`CompressSequenceBegin` returns a unique number that identifies the sequence for subsequent image-compression routines, and it initializes a new image description structure, which is stored in the handle referenced by the `description` local variable.

Using a loop, the `DrawOneFrame` function draws each frame until the last frame is drawn, at which time the function returns the value of `false`. Each frame that it draws is copied to the window so that it can be seen during the compression sequence.

The `CompressSequenceFrame` function is used to compress each frame's image. `CompressSequenceFrame` tells the Image Compression Manager

- which image to compress (in this case, the pixel map of the `currWorld` variable)
- the portion of that image to compress (in this case, all of it)
- whether to update the previous frame's buffer for frame differencing
- the address of the buffer that's to receive the compressed image data

In updating the previous frame's buffer for frame differencing, the Image Compression Manager control flag `codecFlagUpdatePrevious` copies the uncompressed image to the previous frame's buffer; contrast this with the `codecFlagUpdatePreviousComp` flag, which copies the compressed image to the previous frame's buffer—the more lossy the compression, the more the difference between the compressed and uncompressed images.

Image Compression Manager

The `CompressSequenceBegin` function returns a rating of the similarity between the current frame and the previous frame, but this example ignores this rating. After each frame is compressed, the number of bytes in the compressed image data is written to the disk file, followed by the compressed image data itself.

After all the images in the sequence have been compressed, the `CDSequenceEnd` function is called to tell the Image Compression Manager that the sequence is over. The data fork of the file is closed, and the image description is written to a 'SEQU' resource.

The `DrawOneFrame` function draws one frame of the sequence with `QuickDraw`. The frame's image is drawn into the rectangle specified by the `destRect` parameter. The image is a set of **color ramps** in which the shading goes from light to dark in smooth increments. The color ramps fill the destination rectangle and the current frame number centered within the destination rectangle over the ramps.

The `PaintImage` function paints a series of vertical color ramps into the rectangle specified by the `destRect` parameter into the current color graphics port. This is done through a nested loop. The outer loop iterates only twice, and half of the ramps are drawn in the first iteration and half in the second. The inner loop iterates over all the steps in a ramp.

Listing 3-4 Compressing and decompressing a sequence of images: Drawing one frame with `QuickDraw`

```
void CompressSequence (short* dfRef,
                      ImageDescriptionHandle* description)
{
    GWorldPtr      currWorld = nil;
    PixMapHandle    currPixMap;
    CGrafPtr       savedPort;
    GDHandle        savedDevice;
    Handle          buffer = nil;
    Ptr             bufferAddr;
    long            compressedSize;
    long            dataLen;
    Rect            imageRect;
    ImageSequence   sequenceID = 0;
    short           frameNum;
    OSErr           error;
    CodecType       codecKind = 'rle ';

    GetGWorld (&savedPort, &savedDevice);
    imageRect = savedPort->portRect;
    error = NewGWorld (&currWorld, 32, &imageRect, nil, nil, 0);
    CheckError (error, "\pNewGWorld");
    SetGWorld (currWorld, nil);
```

Image Compression Manager

```

currPixMap = currWorld->portPixMap;
LockPixels (currPixMap);

/*
Allocate an embryonic image description structure and the
Image Compression Manager will resize.
*/
*description = (ImageDescriptionHandle) NewHandle (4);

error = CompressSequenceBegin (
    &sequenceID,
    currPixMap,
    nil,                      /* tell ICM to allocate previous
                               image buffer */
    &imageRect,
    &imageRect,
    0,                        /* let ICM choose pixel depth */
    codecKind,
    (CompressorComponent) anyCodec,
    codecNormalQuality, /* spatial quality */
    codecNormalQuality, /* temporal quality */
    5,                        /* at least 1 key frame every
                               5 frames */
    nil,                      /* use default color table */
    codecFlagUpdatePrevious,
    *description );
CheckError (error, "\pCompressSequenceBegin");

error = GetMaxCompressionSize(
    currPixMap,
    &imageRect,
    0,                        /* let ICM choose pixel depth */
    codecNormalQuality, /* spatial quality */
    codecKind,
    (CompressorComponent) anyCodec,
    &compressedSize );
CheckError (error, "\pGetMaxCompressionSize");

buffer = NewHandle(compressedSize);
CheckError (MemError(), "\pNewHandle buffer");
MoveHHi (buffer);
HLock (buffer);
bufferAddr = StripAddress (*buffer);

```

Image Compression Manager

```

for (frameNum = 1; frameNum <= 10; frameNum++)
{
    DrawFrame (&imageRect, frameNum);

    error = CompressSequenceFrame (
        sequenceID,
        currPixMap,
        &imageRect,
        codecFlagUpdatePrevious,
        bufferAddr,
        &compressedSize,
        nil,
        nil );
    CheckError (error, "\pCompressSequenceFrame");

    dataLen = 4;
    error = FSWrite (*dfRef, &dataLen, &compressedSize);
    CheckError (error, "\pFSWrite length");

    error = FSWrite (*dfRef, &compressedSize, bufferAddr);
    CheckError (error, "\pFSWrite buffer");
}

CDSequenceEnd (sequenceID);

DisposeGWorld (currWorld);
SetGWorld (savedPort,savedDevice);
if (buffer) DisposeHandle ( buffer );
}

void DrawFrame (const Rect *imageRect,  long frameNum)
{
    Str255 numStr;

    ForeColor( redColor );
    PaintRect( imageRect );

    ForeColor( blueColor );
    NumToString (frameNum, numStr);
    MoveTo ( imageRect->right / 2, imageRect->bottom / 2);
    TextSize ( imageRect->bottom / 3);
    DrawString (numStr);
}

```

A Sample Function for Decompressing and Playing Back a Sequence From a Disk File

The `SequencePlay` function, shown in Listing 3-5, plays back a sequence of images from a disk file that was created by the `SequenceSave` function (see Listing 3-3 on page 3-40 for details).

The `SequencePlay` function begins by grabbing the image description structure from the file that the user specified from a 'SEQU' resource ID 128. This structure is needed to decompress the images in the file.

Before these compressed images are read, the Image Compression Manager is told to prepare to decompress a sequence of images through the `DecompressSequenceBegin` function. This routine tells the Image Compression Manager

- how the images were compressed with the image description structure
- where to display the decompressed image (the current port in this example)
- what part of the image to decompress (all of it)
- what transfer mode to use when displaying the image (`srcCopy`)
- whether to buffer the image for frame differences

A loop iterates for each frame in the file. For each frame, a long word with the number of bytes in the frame is read from the file, and then that many bytes are read from the file into a compressed-image buffer. This buffer is passed to `DecompressSequenceFrame`, which decompresses the image to the screen (the destination doesn't have to be the screen, but it is in this example). The loop iterates until the end of the file has been reached.

Listing 3-5 Compressing and decompressing a sequence of images: Decompressing and playing back a sequence from a disk file

```
void SequencePlay (void)
{
    ImageDescriptionHandle  description;
    long                   compressedSize;
    Handle                 buffer = nil;
    Ptr                    bufferAddr;
    long                   dataLen;
    long                   lastTicks;
    ImageSequence          sequenceID;
    Rect                   imageRect;
    StandardFileReply      fileReply;
    SFTYPEList             typeList = {'SEQU', 0, 0, 0};
    short                  dfRef = 0;      /* sequence data fork */
    short                  rfRef = 0;      /* sequence resource fork */
    OSErr                  error;
```

Image Compression Manager

```

StandardGetFile (nil, 1, typeList, &fileReply);
if (!fileReply.sfGood) return;

rfRef = FSpOpenResFile (&fileReply.sfFile, fsRdPerm);
CheckError (ResError (), "\pFSpOpenResFile");

description = (ImageDescriptionHandle)
               Get1Resource ('SEQU', 128);
CheckError (ResError (), "\pGet1Resource");

DetachResource ((Handle) description );
HNoPurge ((Handle) description );
CloseResFile (rfRef);

error = FSpOpenDF (&fileReply.sfFile, fsRdPerm, &dfRef);
CheckError (error, "\pFSpOpenDF");

buffer = NewHandle (4);
CheckError (MemError (), "\pNewHandle buffer");

SetRect (&imageRect, 0, 0, (**description).width,
        (**description).height);
error = DecompressSequenceBegin (
    &sequenceID,
    description,
    nil,                /* use the current port */
    nil,                /* go to screen */
    &imageRect,
    nil,                /* no matrix */
    ditherCopy,
    nil,                /* no mask region */
    codecFlagUseImageBuffer,
    codecNormalQuality, /* accuracy */
    (CompressorComponent) anyCodec);

while (true)
{
    dataLen = 4;
    error = FSRead (dfRef, &dataLen, &compressedSize);
    if (error == eofErr)
        break;
    CheckError( error, "\pFSRead" );
}

```

Image Compression Manager

```

if (compressedSize > GetHandleSize (buffer))
{
    HUnlock (buffer);
    SetHandleSize (buffer, compressedSize);
    CheckError (MemError(), "\pSetHandleSize");
}

HLock (buffer);
bufferAddr = StripAddress (*buffer);
error = FSRead (dfRef, &compressedSize, bufferAddr);
CheckError (error, "\pFSRead");

error = DecompressSequenceFrame (
    sequenceID,
    bufferAddr,
    0, // flags
    nil,
    nil );
CheckError (error, "\pDecompressSequenceFrame");

Delay (30, &lastTicks);
}

CDSequenceEnd (sequenceID);
if (dfRef) FSClose (dfRef);
if (buffer) DisposeHandle (buffer);
if (description) DisposeHandle ((Handle)description);
}

```

Spooling Compressed Data

During compression and decompression operations it may be necessary to spool the image data to or from storage other than computer memory. If your application uses the Image Compression Manager functions that handle picture files, the Image Compression Manager manages this spooling for you. However, if you use the functions that work with pixel maps or sequences and your application cannot store the image data in memory, it is your application's responsibility to spool the data.

The Image Compression Manager provides a mechanism that allows the compressors and decompressors to invoke spooling functions provided by your application. There are two kinds of data-spooling functions: data-loading functions and data-unloading functions. Decompressors call data-loading functions during image decompression. The data-loading function is responsible for providing compressed image data to the

decompressor. The decompressor then decompresses the data and writes the resulting image to the appropriate location. See “Application-Defined Functions” beginning on page 3-152 for a detailed description of the calling sequence used by the decompressor component when it invokes your data-loading function.

Compressors call data-unloading functions during image compression. The data-unloading function must remove the compressed image data from memory. The compressor can then compress more of the image and write the compressed image data into the available buffer space. See “Application-Defined Functions” beginning on page 3-152 for a detailed description of the calling sequence used by the compressor component when it invokes your data-unloading function.

When compressing sequences, your application assigns a data-unloading function by calling the `SetCSequenceFlushProc` function (described on page 3-129). When decompressing sequences, you assign a data-loading function by calling the `SetDSequenceDataProc` function (described on page 3-139).

When your application assigns a spooling function to an image or sequence operation, you must also specify a data buffer and the size of that buffer. The `codecMinimumDataSize` value specifies the smallest data buffer you may allocate for image data spooling.

```
#define codecMinimumDataSize 32768      /* minimum data size */
```

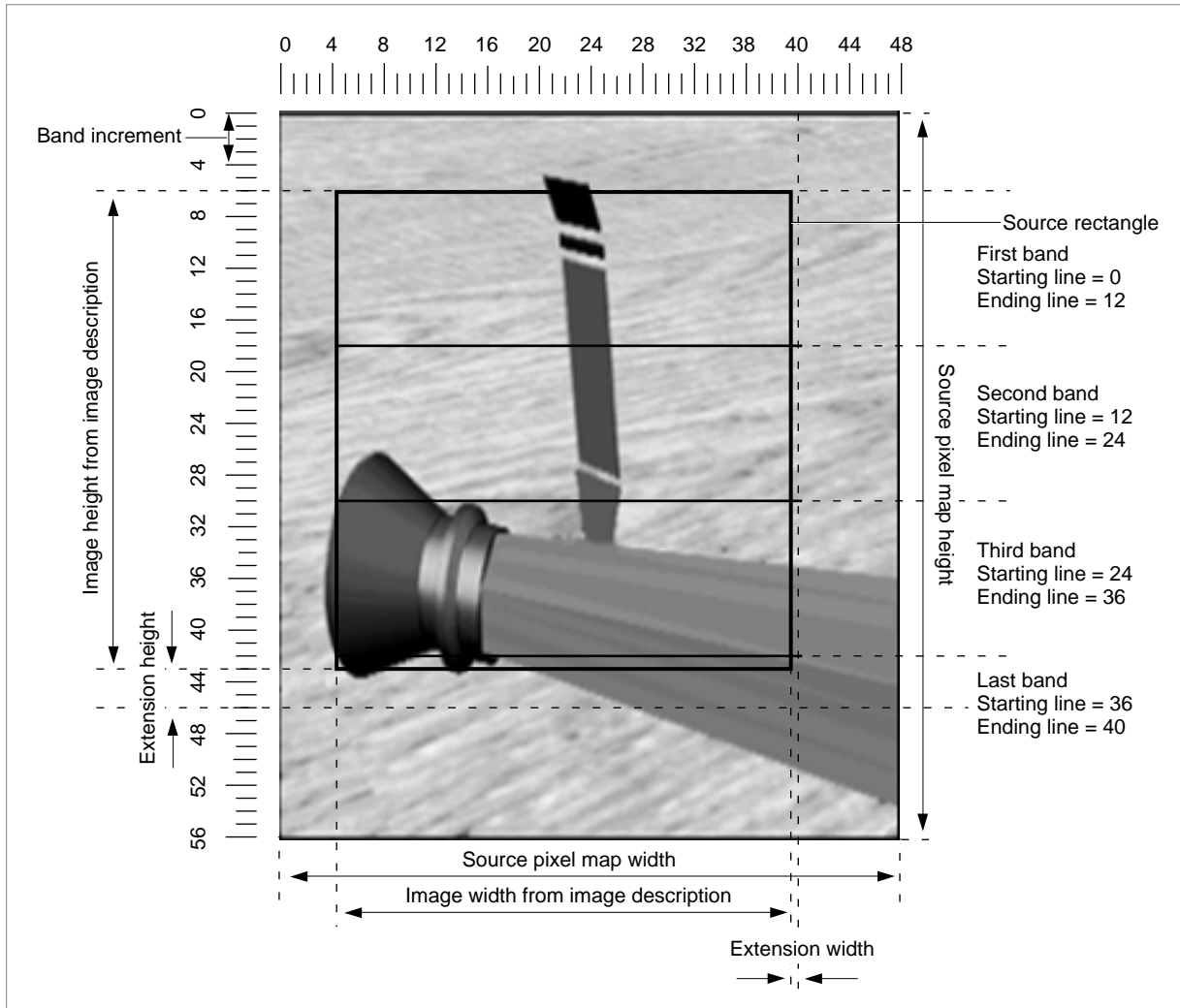
Banding and Extending Images

Occasionally a compressor component may not be able to accommodate the destination rectangle for an image decompression or the source for an image-compression operation. This situation may result from compressors that are optimized to work at certain depths or that cannot perform scaling, translation, dithering, or masking during decompression. In such circumstances the Image Compression Manager allocates a temporary buffer that is acceptable to the compressor component and breaks the image up to fit into that new buffer. Since there often is not enough memory to allocate a buffer to hold the entire image, the Image Compression Manager may allocate one that holds a band of the image. A **band** is one horizontal piece of the image. Its height is some portion of the desired image height (before scaling or rotation), and it is at least as wide as the desired image.

The height of the band is determined both by the amount of memory available and the block size of the compressor component. The block size of a compressor is the natural size at which it handles images, and it is peculiar to the image-compression algorithm. The block size for the photo compressor is usually 16 pixels by 16 pixels, for example. Usually the block width and height are equal, but this is not always the case. The minimum height of a band is one strip of blocks. A strip is defined to be a part of an image that is as high as the block height (for the compressor in question) and as wide as the band. The width of a band is either the width of the desired unscaled image, or that width increased by an extension.

Figure 3-9 shows the measurements of several image bands.

Figure 3-9 Image bands and their measurements



Some compressors can only handle images with dimensions that are a multiple of their block size. If the desired image does not comply with this restriction in either dimension, the Image Compression Manager extends the band on the right side and bottom by the amount required to meet the needs of the compressor. During compression, the compressor fills the extended region with the same pixel value as the pixels adjacent to the extension. During decompression, the Image Compression Manager writes only the pixels that are part of the source image. The extended portion remains only in the offscreen buffer.

Defining Key Frame Rates

The process of temporal compression involves reducing or eliminating temporal redundancy from an image sequence. Temporal compression is most effective when a sequence contains frames that bear significant similarity to adjacent frames. This is typically true of movies and other video sequences. Reconstructing an individual frame within a sequence that has been temporally compressed requires knowledge of the previous frames. This does not present a problem if your application always plays compressed sequences from the beginning. However, if your application needs to start playing a sequence from a random point, or perhaps backward, the decompressor does not have enough information to decompress the frames.

To alleviate this problem, compressors insert key frames in compressed sequences at regular intervals. **Key frames** define starting points for portions of a temporally compressed sequence. Subsequent frames depend on the previous key frame.

At the start of a sequence compression your application can specify a rate at which the compressor is to insert key frames into the compressed data stream. This **key frame rate** indicates the maximum number of frames you will accept between key frames. The Image Compression Manager picks the best key frames from the source sequence and at the same time enforces the specified key frame rate (the best key frames are those that are least similar to adjacent frames, such as at scene changes—these frames would have the largest compressed images even if they were not selected as key frames).

During sequence compression your application can change the key frame rate by calling the `SetCSequenceKeyFrameRate` function (described beginning on page 3-125). By manipulating the parameters for the sequence, you can force the Image Compression Manager to place a key frame at any arbitrary point in a sequence (set the `codecFlagForceKeyFrame` flag to 1 in the `flags` parameter of the `CompressSequenceFrame` function—described beginning on page 3-115).

Fast Dithering

QuickDraw provides a means of displaying images with high color resolution in pixel maps or on screens with lower color resolution. By dithering the destination image, QuickDraw fools your eyes into seeing colors that are not actually available on the display screen. Unfortunately, the error-diffusion technique used by QuickDraw takes longer than just drawing pixels by directly looking them up in a color table. The drawing delays imposed by standard dithering are unacceptable when working with movies.

To alleviate this problem, Apple has developed a technique that allows faster dithering to destinations that use 8 bits per pixel. Fast dithering uses lookup tables created by the Image Compression Manager. All the decompressors supplied by Apple can use fast dithering.

Apple decompressors use fast dithering when copying from image band buffers to 8-bit destinations. If the accuracy for decompression is above normal, then the decompressors use true error diffusion rather than fast dithering. Note that video sequences are normally displayed at normal or low accuracy so that you can obtain maximum display speed during decompression.

Understanding Compressor Components

This section discusses key attributes of compressor components and the functional interfaces these components must support. (**Compressor components** here refers to both image compressor components and image decompressor components.) This information is intended for developers of compressor components. Application developers do not need to be familiar with this material to use the Image Compression Manager.

A compressor component is a code resource that provides image compression or decompression services for image data. These components may also utilize additional hardware to provide their services. Compressor components are registered by the Component Manager, and they present a standard set of function interfaces to the Image Compression Manager (see *Inside Macintosh: QuickTime Components* for a detailed description of the functions that compressors must provide). A compressor can be a systemwide resource, or it can be local to a particular application.

Applications never communicate directly with compressors. Applications request compressor services by issuing the appropriate Image Compression Manager functions. The Image Compression Manager then performs its necessary processing before invoking the compressor. Of course, an application could install its own compressor component. However, any interaction between the application and the compressor is still managed by the Image Compression Manager.

The Image Compression Manager knows about two types of compressor components. Components that can compress image data carry a component type (described by the `compressorComponentType` data type) of 'imco' and are referred to as *compressors*. Components that can decompress images have a component type (described by the `decompressorComponentType` data type) of 'imdc' and are called *decompressors*. The value of the component subtype indicates the compression algorithm supported by the component. All compressor components with the same subtype must be able to handle the same format of compressed data. During decompression a component should handle all variations of the data specified for a subtype. Conversely, while compressing an image a compressor must not produce data that decompressors of the same subtype cannot handle during decompression.

The Image Compression Manager defines four callback functions that may be provided to compressors or decompressors by applications. A **callback function** is an application-defined function that is invoked at a specified time or based on specified criteria. These callback functions are data-loading functions, data-unloading functions, completion functions, and progress functions. Data-loading functions and data-unloading functions support spooling of compressed data. Completion functions allow compressors and decompressors to report that asynchronous operations have completed. Progress functions provide a mechanism for compressors and decompressors to report their progress toward completing an operation. For more information about these callback functions, see "Application-Defined Functions" beginning on page 3-152.

Image Compression Manager Reference

This section describes all of the Image Compression Manager functions and data structures. The Image Compression Manager provides a rich and varied set of functions that allow your application to work with compressed image data. This discussion has been divided into the following sections:

- “Data Types” identifies the data structures used by your application when interacting with the Image Compression Manager.
- “Image Compression Manager Functions” describes the functions that your application can use to work with compressed data.
- “Application-Defined Functions” describes the interfaces to the callback functions that may be provided to compressors or decompressors by applications.

Data Types

This section describes the format and content of the data structures, data types, and constants that you use to exchange information with the Image Compression Manager.

The Image Description Structure

An image description structure contains information that defines the characteristics of a compressed image or sequence. Data in the image description structure indicates the type of compression that was used, the size of the image when displayed, the resolution at which the image was captured, and so on. One image description structure may be associated with one or more compressed frames.

The `ImageDescription` data type defines the layout of an image description structure. In addition, an image description structure may contain additional data in extensions and custom color tables. The Image Compression Manager provides functions that allow you to get and set the data in image description structure extensions and custom color tables.

- See “Working With Images,” which begins on page 3-77, for more information about the functions `GetImageDescriptionCTable` and `SetImageDescriptionCTable`, which allow you to work with custom color tables in image description structures.
- See *Inside Macintosh: QuickTime Components* for more information about the `GetImageDescriptionExtension`, `SetImageDescriptionExtension`, `RemoveImageDescriptionExtension`, `CountImageDescriptionExtensionType`, and `GetNextImageDescriptionExtensionType` functions, which allow you to work with image description structure extensions.

Image Compression Manager

```

struct ImageDescription {
    long idSize;          /* total size of this structure */
    CodecType cType;      /* compressor creator type */
    long resvd1;          /* reserved--must be set to 0 */
    short resvd2;         /* reserved--must be set to 0 */
    short dataRefIndex;   /* reserved--must be set to 0 */
    short version;        /* version of compressed data */
    short revisionLevel;  /* compressor that created data */
    long vendor;          /* compressor developer that created data */
    CodecQ temporalQuality;
                        /* degree of temporal compression */
    CodecQ spatialQuality;
                        /* degree of spatial compression */
    short width;          /* width of source image in pixels */
    short height;         /* height of source image in pixels */
    Fixed hRes;           /* horizontal resolution of source image */
    Fixed vRes;           /* vertical resolution of source image */
    long dataSize;        /* size in bytes of compressed data */
    short frameCount;     /* number of frames in image data */
    Str31 name;           /* name of compression algorithm */
    short depth;          /* pixel depth of source image */
    short clutID;         /* ID number of the color table for image */
};
typedef struct ImageDescription ImageDescription;
typedef ImageDescription *ImageDescriptionPtr,
**ImageDescriptionHandle;

```

Field descriptions

idSize	Defines the total size of this image description structure with extra data including color lookup tables and other per sequence data.
cType	Indicates the type of compressor component that created this compressed image data. The value of this field indicates the compression algorithm supported by the component. The <code>Codec</code> data type defines a field in the compressor name list structure that identifies the compression method employed by a given compressor component. Apple Computer's Developer Technical Support group assigns these values so that they remain unique. These values correspond, in turn, to text strings that can identify the compression method to the user. See the description of <code>GetCodecNameList</code> on page 3-67 for a list of valid values.
resvd1	Reserved for Apple. This field must be set to 0.
resvd2	Reserved for Apple. This field must be set to 0.
dataRefIndex	Reserved for Apple. This field must be set to 0.
version	Indicates the version of the compressed data. The contents of this field should indicate the version of the compression algorithm that

Image Compression Manager

	was used to create the compressed data. By examining this field, decompressors that support many versions of an algorithm can determine the proper way to decompress the image.
revisionLevel	Indicates the version of the compressor that created the compressed image. Developers of compressors and decompressors assign these version numbers.
vendor	Identifies the developer of the compressor that created the compressed image.
temporalQuality	Indicates the degree of temporal compression performed on the image data associated with this description. This field is valid only for sequences. See “Compression Quality Constants” beginning on page 3-61 for a list of available values.
spatialQuality	Indicates the degree of spatial compression performed on the image data associated with this description. This field is valid for sequences and still images. See “Compression Quality Constants” on page 3-61 for a list of available values.
width	Contains the width of the source image, in pixels.
height	Contains the height of the source image, in pixels.
hRes	Contains the horizontal resolution of the source image, in dots per inch.
vRes	Contains the vertical resolution of the source image, in dots per inch.
dataSize	Indicates the size of the compressed image, in bytes. This field is valid only for still images. Set this field to 0 if the size is unknown.
frameCount	Contains the number of frames in the image data associated with this description.
name	Indicates the compression algorithm used to create the compressed data. This algorithm is stored in Pascal string format. It always takes up 32 bytes no matter how long the string is. The 32 bytes consist of 31 bytes plus one length byte. The value of this field should correspond to the compressor type specified by the <code>cType</code> field, as well as to the value of the <code>typeName</code> field in the appropriate compressor name structure returned by the <code>GetCodecNameList</code> function (see “The Compressor Name List Structure” on page 3-60 for information on the compressor list name structure; see “Getting Information About Compressor Components,” which begins on page 3-66, for information on the <code>GetCodecNameList</code> function). Applications may use the contents of this field to indicate the type of compression used for the associated image.
depth	Contains the pixel depth specified for the compressed image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the depth of color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

Image Compression Manager

`clutID` Contains the ID of the color table for the compressed image, or other special values. If this field is set to 0, then a custom color table is defined for the compressed image. You can use the `GetImageDescriptionCTable` function, described on page 3-91, to retrieve the color table. If this field is set to -1, the image does not use a color table.

The Compressor Information Structure

Your application can retrieve information describing the capabilities of compressors with the `GetCodecInfo` function (described on page 3-69). The `CodecInfo` data type defines the format of the compressor information structure.

```
/* compressor information structure */
struct CodecInfo {
    Str31 typeName;          /* compression algorithm (codec type) */
    short version;           /* version supported by component */
    short revisionLevel;     /* version assigned by developer */
    long vendor;             /* developer of component */
    long decompressFlags;    /* decompression capability flags */
    long compressFlags;      /* compression capability flags */
    long formatFlags;        /* compression format flags */
    unsigned char compressionAccuracy;
                            /* relative accuracy of this algorithm */
    unsigned char decompressionAccuracy;
                            /* relative accuracy of this algorithm */
    unsigned short compressionSpeed;
                            /* relative compression speed */
    unsigned short decompressionSpeed;
                            /* relative decompression speed */
    unsigned char compressionLevel;
                            /* relative compression of component */
    char resvd;              /* reserved--set to 0 */
    short minimumHeight;     /* minimum image height for component */
    short minimumWidth;      /* minimum image width for component */
    short decompressPipelineLatency;
                            /* in milliseconds (asynchronous) */
    short compressPipelineLatency;
                            /* in milliseconds (asynchronous) */
    long privateData;        /* reserved for use by Apple */
};
typedef struct CodecInfo CodecInfo;
```


Image Compression Manager

Field descriptions

<code>typeName</code>	Indicates the compression algorithm used by the component—for example, 'Animation'. This Pascal string may be used to identify the compression algorithm to the user. The string always takes up 32 bytes no matter how long it is. The 32 bytes consist of 31 bytes plus one length byte. Apple Computer's Developer Technical Support group assigns these type names. The value of this field should correspond to the value of the <code>typeName</code> field in the appropriate compressor name structure returned by the <code>GetCodecNameList</code> function (see "The Compressor Name Structure" on page 3-59 for information on the compressor name structure; see page 3-67 for information on the <code>GetCodecNameList</code> function).
<code>version</code>	Indicates the version of compressed data this component supports. The contents of this field should indicate the most recent version of the compression algorithm that the component can understand.
<code>revisionLevel</code>	Indicates the version of the component—for example, 0x00010001 (1.0.1). Developers of compressors assign these version numbers.
<code>vendor</code>	Identifies the developer of the component—for example, 'appl'. The value of this field corresponds to the manufacturer code or application signature assigned to the developer.
<code>decompressFlags</code>	Contains flags that specify the decompression capabilities of the component. Typically, these flags are of interest only to developers of image decompressors. The bit values for this field are described in the discussion of image decompressors in <i>Inside Macintosh: QuickTime Components</i> .
<code>compressFlags</code>	Contains flags that specify the compression capabilities of the component. Typically, these flags are of interest only to developers of image compressors. The bit values for this field are described in the discussion of image compressors in <i>Inside Macintosh: QuickTime Components</i> .
<code>formatFlags</code>	Contains flags that describe the possible format for compressed data produced by this component and the format of compressed files that the component can handle during decompression. Typically, these flags are of interest only to developers of compressor components. The bit values for this field are described in the discussion of image compressor and decompressor components in <i>Inside Macintosh: QuickTime Components</i> .
<code>compressionAccuracy</code>	Indicates the relative accuracy of the compression algorithm employed by the component. Valid values for this field range from 0 to 255. A value of 0 means that the accuracy is unknown. Values from 1 to 255 provide a gauge for the relative accuracy of the compression algorithm—higher values indicate better accuracy.

Image Compression Manager

The Image Compression Manager examines this field to determine which compressor component can most accurately compress a given image.

The `compressionAccuracy` field can only approximate the accuracy of a compression algorithm. Typically, compression algorithms produce results of varying quality based on a variety of parameters, including image size and content. Since this information is not available until a compression request is issued, a precise measure of accuracy is not possible. However, the value of this field should still give a rough idea of the accuracy of the supported algorithm.

`decompressionAccuracy`

Indicates the relative accuracy of the decompression algorithm employed by the component. Valid values for this field range from 0 to 255. A value of 0 means that the accuracy is unknown. Values from 1 to 255 indicate the relative accuracy of the decompression technique—higher values mean better accuracy.

The Image Compression Manager examines this field to determine which decompressor component can most accurately decompress a given image.

The `decompressionAccuracy` field can only approximate the accuracy of a decompression algorithm. Typically, decompression algorithms produce results of varying quality based on a variety of parameters, including image size and content. Since this information is not available until a decompression request is issued, a precise measure of accuracy is not possible. However, the value of this field should still give a rough idea of the accuracy of the supported algorithm.

`compressionSpeed`

Indicates the relative speed of the component for compression operations. Valid values for this field lie in the range from 0 to 65,535. A value of 0 means that the speed is unknown. Values from 1 to 65,535 correspond to the number of milliseconds the component requires to compress a 320-by-240 pixel image on a Macintosh II computer.

The Image Compression Manager examines this field to determine which compressor component can most quickly compress a given image.

`decompressionSpeed`

Indicates the relative speed of the component for decompression operations. Valid values for this field lie in the range from 0 to 65,535. A value of 0 means that the speed is unknown. Values from 1 to 65,535 correspond to the number of milliseconds the component requires to decompress a 320-by-240 pixel image on a Macintosh II computer.

The Image Compression Manager examines this field to determine which compressor component can most quickly decompress a given image.

Image Compression Manager

`compressionLevel`

Indicates the relative compression achieved by this component. Valid values for this field lie in the range from 0 to 255. A value of 0 means that the compression level is unknown. Values from 1 to 255 map to percentage values of relative compression—lower values mean lesser compression. A value of 1 means no compression (0 percent); a value of 255 means maximum compression (100 percent).

The Image Compression Manager examines this field to determine which available compressor component will yield the smallest resulting data for a given image.

The `compressionLevel` field can only approximate the effectiveness of a compression algorithm. Typically, compression algorithms produce results of varying quality based on a variety of parameters, including image size and content. Since this information is not available until a compression request is issued, a precise measure of compression is not possible. However, the value of this field should still give a rough idea of the effectiveness of the supported algorithm.

`resvd`

Reserved for Apple. This field must be set to 0.

`minimumHeight`

Specifies the height in pixels of the smallest image the component can handle. Together with the `minimumWidth` field, this field defines the block size for the component. The Image Compression Manager does not issue compression or decompression requests for images smaller than the block size.

`minimumWidth`

Specifies the width in pixels of the smallest image the component can handle. Together with the `minimumHeight` field, this field defines the block size for the component. The Image Compression Manager does not issue compression or decompression requests for images smaller than the block size.

`decompressPipelineLatency`

Reserved for future use. This field must be set to 0.

`compressPipelineLatency`

Reserved for future use. This field must be set to 0.

`privateData`

Reserved for use by Apple. This field must be set to 0.

The Compressor Name Structure

The `CodecNameSpec` data type defines a compressor name structure.

```
/* compressor name structure from GetCodecNameList function */
struct CodecNameSpec
{
    CodecComponent codec; /* component ID for compressor */
    CodecType cType;      /* type identifier for compressor */
    Str31 typeName;       /* string identifier of algorithm */
}
```

Image Compression Manager

```

    Handle name;          /* name of compressor component */
};
typedef struct CodecNameSpec CodecNameSpec;

```

Field descriptions

<code>codec</code>	Uniquely identifies the component or, in some cases, contains a special value that selects all components. If your application requests a list of components, the <code>codec</code> field in each compressor name structure contains the component ID for that compressor. If your application requests a list of component types, the <code>codec</code> field is set to 0 in each compressor name structure.
<code>cType</code>	Contains the type identifier for the compressor. The value of this field indicates the compression algorithm supported by the component. See the description of <code>GetCodecNameList</code> on page 3-67 for a list of valid values.
<code>typeName</code>	Contains a text string in Pascal format that identifies the compression algorithm supported by the component. This string may be used to identify the compression algorithm to the user. The value of this field should correspond to the value of the <code>typeName</code> field in the appropriate compressor information structure returned by the component in response to a <code>GetCodecInfo</code> function (see “The Compressor Information Structure” on page 3-56 for information on the compressor information structure; see page 3-69 for information on the <code>GetCodecInfo</code> function).
<code>name</code>	Specifies the name of the compressor component. Developers assign these names to uniquely identify their products. This name may be used to identify the component to the user.

The Compressor Name List Structure

The compressor name list structure contains a list of compressor name structures. (A compressor name structure identifies a compressor or decompressor component.) The data structure contains name and type information for the component. The `GetCodecNameList` function returns an array of these structures, formatted into a compressor name list structure. See page 3-67 for more information on the `GetCodecNameList` function. The `CodecNameSpecList` data type defines a compressor name list structure.

```

/* compressor name list structure */
struct CodecNameSpecList {
    short count;    /* how many compressor name structures */
    CodecNameSpec list[1];
                  /* array of compressor name structures */
};
typedef struct CodecNameSpecList CodecNameSpecList;
typedef CodecNameSpecList *CodecNameSpecListPtr;

```

Field descriptions

<code>count</code>	Indicates the number of compressor name structures contained in the <code>list</code> array that follows.
<code>list</code>	Contains an array of compressor name structures. Each structure corresponds to one compressor component or type that meets the selection criteria your application specifies when it issues the <code>GetCodecNameList</code> function. The <code>count</code> field indicates the number of structures stored in this array.

Compression Quality Constants

Compressor components may allow applications to assert some control over the image quality that results from a compression or decompression operation. For example, the `CompressSequenceBegin` function (described on page 3-110) provides the `spatialQuality` and `temporalQuality` parameters so that applications can indicate the level of image accuracy desired within individual frames and across adjacent frames in a sequence, respectively. These quality values become a property of the compressed data and are stored in the image description structure (described on page 3-53) associated with the image or sequence.

For a given compression operation, your application can determine the quality that the component supports by issuing the `GetCompressionTime` function (described on page 3-73).

The `CodecQ` data type defines a field that identifies the quality characteristics of a given image or sequence. Note that individual components may not implement all the quality levels shown here. In addition, components may implement other quality levels in the range from `codecMinQuality` to `codecMaxQuality`. Relative quality should scale within the defined value range. Values above `codecLosslessQuality` are reserved for use by individual components.

```
/* compression quality values */
#define codecMinQuality 0x000L /* minimum valid value */
#define codecLowQuality 0x100L /* low-quality reproduction */
#define codecNormalQuality
                                0x200L /* normal-quality repro */
#define codecHighQuality
                                0x300L /* high-quality repro */
#define codecMaxQuality 0x3FFL /* maximum-quality repro */
#define codecLosslessQuality
                                0x400L /* lossless-quality repro */
typedef unsigned long CodecQ;
```

Image Compression Manager

Constant descriptions`codecMinQuality`Specifies the minimum valid value for a `CodecQ` field.`codecLowQuality`

Specifies low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

`codecNormalQuality`

Specifies image reproduction of normal quality.

`codecHighQuality`

Specifies high-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.

`codecMaxQuality`Specifies the maximum standard value for a `CodecQ` field.`codecLosslessQuality`

Specifies lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Image Compression Manager Function Control Flags

A number of Image Compression Manager functions take control flags that allow your application to exert greater control over the operation. In some cases, the Image Compression Manager returns status information about the results of the function in the same flags field. In general, you need to use only a few of these flags. The function descriptions in the reference section of this chapter indicate the flags that are valid for individual functions.

The `CodecFlags` data type defines these flag fields.

```
typedef unsigned short CodecFlags;

/* Image Compression Manager function control flags */
#define codecFlagUseImageBuffer (1L<<0)
                                /* (input) use image buffer */
#define codecFlagUseScreenBuffer (1L<<1)
                                /* (input) use screen buffer */
#define codecFlagUpdatePrevious (1L<<2)
                                /* (input) update previous buffer */
#define codecFlagNoScreenUpdate (1L<<3)
                                /* (input) don't update screen */
#define codecFlagWasCompressed (1L<<4)
                                /* (input) image was compressed */
#define codecFlagDontOffscreen (1L<<5)
                                /* don't go offscreen */
```

Image Compression Manager

```

#define codecFlagUpdatePreviousComp (1L<<6)
                                /* (input) update previous buffer */
#define codecFlagForceKeyFrame (1L<<7)
                                /* force key frame from image */
#define codecFlagOnlyScreenUpdate
                                (1L<<8)
                                /* (input) only update screen */
#define codecFlagLiveGrab (1L<<9)
                                /* (input) grab live video */
#define codecFlagUsedNewImageBuffer
                                (1L<<14)
                                /* (output) new image buffer used */
#define codecFlagUsedImageBuffer
                                (1L<<15)
                                /* (output) decompressor used
                                offscreen buffer */

```

Constant descriptions**codecFlagUseImageBuffer**

Controls whether the decompressor allocates an offscreen buffer for decompression. If your application sets this flag to 1, the decompressor allocates an offscreen buffer the size of the compressed image. If you set this flag to 0, the decompressor does not use an offscreen image buffer. These image buffers are useful when decompressing sequences that were created using temporal compression. For more information about image buffers, see “Using Screen Buffers and Image Buffers” on page 3-38.

codecFlagUseScreenBuffer

Controls whether the decompressor allocates an offscreen destination buffer during decompression. If you set this flag to 1, the decompressor allocates an offscreen buffer the size of the destination screen. If you set this flag to 0, the decompressor does not use an offscreen screen buffer. Using a screen buffer helps to reduce tearing that may result when decompressing directly to the screen. For more information about screen buffers, see “Using Screen Buffers and Image Buffers” on page 3-38.

codecFlagUpdatePrevious

Controls whether the compressor updates the previous image buffer during compression. This flag is only used with sequences that are being temporally compressed. If you set this flag to 1, the compressor copies the current source image into the previous frame buffer at the end of the frame compression.

Image Compression Manager

`codecFlagNoScreenUpdate`

Controls whether the decompressor updates the screen image. If you set this flag to 1, the decompressor does not write the current frame to the screen, but does write the frame to its offscreen image buffer (if one was allocated). If you set this flag to 0, the decompressor writes the frame to the screen.

`codecFlagWasCompressed`

Indicates to the compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. Set this flag to 1 to indicate that the image was previously compressed. Set this flag to 0 if the image was not previously compressed.

`codecFlagDontOffscreen`

Controls whether the decompressor uses the offscreen buffer during sequence decompression. This flag is only used with sequences that have been temporally compressed. If this flag is set to 1, the decompressor does not use the offscreen buffer during decompression. Instead, the decompressor returns an error. This allows your application to refill the offscreen buffer. If this flag is set to 0, the decompressor uses the offscreen buffer if appropriate.

`codecFlagUpdatePreviousComp`

Controls whether the compressor updates the previous image buffer with the decompressed image data. This flag is only used with temporal compression and is similar to the `codecFlagUpdatePrevious` flag. As with the `codecFlagUpdatePrevious` flag, if you set this flag to 1, the compressor updates the previous frame buffer at the end of the frame compression. However, this flag causes the Image Compression Manager to update the frame buffer using an image obtained by decompressing the results of the most recent compression operation, rather than the source image.

`codecFlagForceKeyFrame`

Controls whether the compressor creates a key frame from the current image. This flag is only used with temporal compression. If you set this flag to 1, the compressor makes the current image a key frame. If you set this flag to 0, the compressor decides based on other criteria, such as the key frame rate, whether to create a key frame from the current image.

`codecFlagOnlyScreenUpdate`

Controls whether the decompressor decompresses the current frame. If you set this flag to 1, the decompressor writes the contents of its offscreen image buffer to the screen, but does not decompress the current frame. If you set this flag to 0, the decompressor decompresses the current frame and writes it to the screen. You can set this flag to 1 only if you have allocated an offscreen image buffer for use by the decompressor.

Image Compression Manager

`codecFlagLiveGrab`

Indicates to the compressor whether the current sequence results from grabbing live video. When working with live video, compressors operate as quickly as possible and disable some additional processing, such as compensation for previously compressed data. Set this flag to 1 when you are compressing from a live video source—the compressor then operates as quickly as it can.

`codecFlagUsedNewImageBuffer`

Indicates to your application that the decompressor used the offscreen image buffer for the first time when it processed this frame. If this flag is set to 1, the decompressor used the image buffer for this frame and this is the first time the decompressor used the image buffer in this sequence. If this flag is set to 0, the decompressor did not use the image buffer.

`codecFlagUsedImageBuffer`

Indicates to your application that the decompressor used the offscreen image buffer for this frame. If this flag is set to 1, the decompressor used the image buffer. If this flag is set to 0, the decompressor did not use the image buffer.

Image Compression Manager Functions

The following sections describe the functions that the Image Compression Manager provides to application programs. This section is divided into the following topics:

- “Getting Information About Compressor Components” describes the Image Compression Manager functions that allow applications to gather information about the manager and installed compressor components
- “Getting Information About Compressed Data” describes the functions that allow applications to obtain information about compressed images
- “Working With Images” defines the functions that applications can use to compress and decompress single-frame images that are stored in pixel maps
- “Working With Pictures and PICT Files” describes the functions that applications can use to compress and decompress single-frame images that are stored as pictures or picture files (PICT files)
- “Making Thumbnail Pictures” defines the functions that create and manipulate thumbnail images
- “Working With Sequences” describes the functions that allow applications to compress and decompress sequences of images
- “Changing Sequence-Compression Parameters” discusses the functions that your application can use to manipulate many of the parameters that govern sequence-compression operations
- “Constraining Compressed Data” describes the functions and a data structure that allow your application to communicate information to compressors that can constrain compressed data to a specific data rate

Image Compression Manager

- “Changing Sequence-Decompression Parameters” discusses the functions that your application can use to manipulate many of the parameters that govern sequence-decompression operations
- “Working With the StdPix Function” describes the functions that work with the StdPix function to allow your application to have access to compressed image data as it is displayed
- “Aligning Windows” describes the functions that your application can use to position individual windows along optimal alignment grids
- “Working With Graphics Devices and Graphics Worlds” discusses the functions that let you select graphics devices and specify graphics worlds for use in compression and decompression operations

Getting Information About Compressor Components

This section describes the functions that allow your application to gather information about the Image Compression Manager and the installed compressor components.

You can use the `CodecManagerVersion` function to retrieve the version number associated with the Image Compression Manager that is installed on a particular computer.

You can use the `FindCodec`, `GetCodecInfo`, and `GetCodecNameList` functions to locate and retrieve information about the compressor components that are available on a computer.

CodecManagerVersion

Your application can determine the version of the installed Image Compression Manager by calling the `CodecManagerVersion` function.

```
pascal OSErr CodecManagerVersion (long *version);
```

version Contains a pointer to a long integer that is to receive the version information. The Image Compression Manager returns its version number into this location. The version number is a long integer value.

DESCRIPTION

The `CodecManagerVersion` function returns the version information as a long integer value.

RESULT CODES

noErr 0 No error

SEE ALSO

“Getting Information About Compressors and Compressed Data,” which begins on page 3-28, describes how to use `CodecManagerVersion`.

GetCodecNameList

The `GetCodecNameList` function allows your application to retrieve a list of installed compressor components or a list of installed compressor types. This information may be useful when the user selects a compression type for a given image or sequence.

```
pascal OSErr GetCodecNameList (CodecNameSpecListPtr *list,
                               short showAll);
```

<code>list</code>	Contains a pointer to a field that is to receive a pointer to the compressor name list structure. The Image Compression Manager creates the appropriate list and returns a pointer to that list in the field specified by the <code>list</code> parameter. Note that the <code>GetCodecNameList</code> function creates this list in your application's current heap zone.
<code>showAll</code>	Specifies a short integer that controls the contents of the list. Set this parameter to 1 to receive a list of the names of all installed compressor components—the returned list contains one entry for each installed compressor. Set this parameter to 0 to receive a list of the types of installed compressor components—the returned list contains one entry for each installed compressor type. See “The Compressor Name List Structure” on page 3-60 for a complete description of the contents of the returned list.

DESCRIPTION

The Image Compression Manager returns this information in a compressor name list structure, which contains an array of compressor name structures and a field indicating the number of structures in the array.

The `CodecType` data type defines a field in the compressor name list structure that identifies the compression method employed by a given compressor component. Apple Computer's Developer Technical Support group assigns these values so that they remain unique. These values correspond, in turn, to text strings that can identify the compression method to the user.

```
typedef long CodecType;
/* compressor type descriptor--for example 'jpeg', 'rle ',
   'rpza' */
```

Image Compression Manager

Currently, six `CodecType` values are provided by Apple. You should use the `GetCodecNameList` function to retrieve these names, so that your application can take advantage of new compressor types that may be added in the future. For each `CodecType` value in the following list, the corresponding compression method is also identified by its text string name. For more information about each of these compression techniques, see the section “About Image Compression,” which begins on page 3-8.

Table 3-3 Compressor type descriptors

Compressor type	Compressor name
'rpza'	video compressor
'jpeg'	photo compressor
'rle '	animation compressor
'raw '	raw compressor
'smc '	graphics compressor
'cdvc'	compact video compressor

SPECIAL CONSIDERATIONS

Note that the Image Compression Manager returns the list in your application's current heap zone. Use the `DisposeCodecNameList` function, described in the next section, to release this memory when your program is finished with the list.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified

Component Manager errors
Memory Manager errors

DisposeCodecNameList

The `DisposeCodecNameList` function allows your application to dispose of the compressor name list structure you obtained by calling the `GetCodecNameList` function.

```
pascal OSErr DisposeCodecNameList (CodecNameSpecListPtr list);
```

Image Compression Manager

`list` Points to the compressor name list to be disposed of. You obtain the compressor list by calling the `GetCodecNameList` function, which is described in the previous section.

RESULT CODES

`noErr` 0 No error
Memory Manager errors

GetCodecInfo

The `GetCodecInfo` function returns information about a single compressor component.

```
pascal OSErr GetCodecInfo (CodecInfo *info, CodecType cType,  
                           CodecComponent codec);
```

`info` Contains a pointer to a compressor information structure. The `GetCodecInfo` function returns the detailed information about the appropriate compressor component into this structure.

`cType` Specifies a compressor type. You must set this parameter to a valid compressor type (see Table 3-3 on page 3-68 for a list of the available compressor types).

`codec` Specifies a compressor identifier. Set this parameter to the component identifier of the specific compressor for the request. The component identifier is available in the compressor name list structure returned by the `GetCodecNameList` function (described on page 3-67).

If you want information about any compressor of the type specified by the `cType` parameter, set `codec` to 0. The Image Compression Manager then returns information about the first compressor it finds of the type you have specified.

DESCRIPTION

Your application may retrieve information about a specific compressor or about a compressor of a specific type. If you request information about a type of compressor, the Image Compression Manager returns information about the first compressor it finds of that type. The Image Compression Manager returns the detailed compressor information in a compressor information structure (see “The Compressor Information Structure,” which begins on page 3-56, for details).

Image Compression Manager

RESULT CODES

<code>noErr</code>	0	No error
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor

Component Manager errors

Memory Manager errors

FindCodec

The `FindCodec` function allows you to determine which of the installed compressors or decompressors has been chosen to field requests made using one of the special compressor identifiers.

Some Image Compression Manager functions allow you to specify a particular compressor component. For example, you may use the `codec` parameter to the `CompressSequenceBegin` function (described on page 3-110) to specify a particular compressor to do the compression.

You identify the compressor to the Image Compression Manager by specifying the compressor's component identifier (see the description of the `GetCodecNameList` function on page 3-67 for information on retrieving these identifiers).

The Image Compression Manager also supports several special identifiers that allow you to exert some control over the component for a given action without having to know its identifier.

```
pascal OSErr FindCodec (CodecType cType, CodecComponent specCodec,
                        CompressorComponent *compressor,
                        DecompressorComponent *decompressor);
```

`cType` Specifies a compressor type. You must set this parameter to a valid compressor type (see Table 3-3 on page 3-68 for a list of the available compressor types).

`specCodec` Contains a special identifier value. You must set this parameter to one of the following special identifier values:

`anyCodec` Choose the first compressor or decompressor of the specified type

`bestSpeedCodec` Choose the fastest compressor or decompressor of the specified type

`bestFidelityCodec` Choose the most accurate compressor or decompressor of the specified type

`bestCompressionCodec` Choose the compressor that produces the smallest resulting data

Image Compression Manager

compressor

Contains a pointer to a field to receive the identifier for the compressor component. The Image Compression Manager returns the identifier of the compressor that meets the special characteristics you specify in the `specCodec` parameter. Note that this identifier may differ from the value of the field referred to by the `decompressor` field. The Image Compression Manager sets this field to 0 if it cannot find a suitable compressor component. Set this parameter to `nil` if you do not want this information.

decompressor

Contains a pointer to a field to receive the identifier for the decompressor component. The Image Compression Manager returns the identifier of the decompressor that meets the special characteristics you specify in the `specCodec` parameter. Note that this identifier may differ from the value of the field referred to by the `compressor` field. The Image Compression Manager sets this field to 0 if it cannot find a suitable decompressor component. Set this parameter to `nil` if you do not want this information.

DESCRIPTION

You can use the `FindCodec` function to obtain the identifier of the component that is being used to field requests made with one of the special compressor identifiers.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>memFullErr</code>	-108	Not enough memory available
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor

Getting Information About Compressed Data

This section describes the functions that enable your application to collect information about compressed images and images that are about to be compressed. Your application may use some of these functions in preparation for compressing or decompressing an image or sequence.

You can use the `GetCompressionTime` function to determine how long it will take for a compressor to compress a specified image. Similarly, you can use the `GetMaxCompressionSize` function to find out how large the compressed image may be after the compression operation.

You can use the `GetCompressedImageSize` to determine the size of a compressed image that does not have a complete image description.

The `GetSimilarity` function allows you to determine how similar two images are. This information is useful when you are performing temporal compression on an image sequence.

GetMaxCompressionSize

The `GetMaxCompressionSize` function allows your application to determine the maximum size an image will be after compression. You specify the compression characteristics, including compression type and quality, along with the image.

```
pascal OSErr GetMaxCompressionSize (PixMapHandle src,
                                     const Rect *srcRect,
                                     short colorDepth,
                                     CodecQ quality,
                                     CodecType cType,
                                     CompressorComponent codec,
                                     long *size);
```

<code>src</code>	Contains a handle to the source image. The source image must be stored in a pixel map structure. The compressor uses only the image's size and pixel depth to determine the maximum size of the compressed image.
<code>srcRect</code>	Contains a pointer to a rectangle defining the portion of the source image that is to be compressed. You may set this parameter to <code>nil</code> if you are interested only in information about quality settings. <code>GetCompressionTime</code> then uses the bounds of the source pixel map.
<code>colorDepth</code>	Specifies the depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the <code>GetCodecInfo</code> function (see "Getting Information About Compressor Components" on page 3-66 for more information on the <code>GetCodecInfo</code> function).
<code>quality</code>	Specifies the desired compressed image quality. See "Compression Quality Constants" beginning on page 3-61 for valid values.
<code>cType</code>	Specifies a compressor type. You must set this parameter to a valid compressor type (see Table 3-3 on page 3-68 for a list of the available compressor types).
<code>codec</code>	Contains a compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use one of the special identifiers: <ul style="list-style-type: none"> <code>anyCodec</code> Choose the first compressor of the specified type <code>bestSpeedCodec</code> Choose the fastest compressor of the specified type <code>bestFidelityCodec</code> Choose the most accurate compressor of the specified type

	<code>bestCompressionCodec</code>	Choose the compressor that produces the smallest resulting data
		You can also specify a component instance. This may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation.
<code>size</code>		Contains a pointer to a field to receive the size, in bytes, of the compressed image.

DESCRIPTION

The Image Compression Manager returns the maximum resulting size for the specified image and parameters. Your application may then use this information to allocate memory for the compression operation.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor

GetCompressionTime

The `GetCompressionTime` function allows your application to determine the estimated amount of time required to compress a given image. This function also allows you to verify that the quality settings you desire are supported by a given compressor component.

You specify the compression characteristics, including compression type and quality, along with the image.

```
pascal OSErr GetCompressionTime (PixMapHandle src,
                                const Rect *srcRect,
                                short colorDepth,
                                CodecType cType,
                                CompressorComponent codec,
                                CodecQ *spatialQuality,
                                CodecQ *temporalQuality,
                                unsigned long *compressTime);
```

<code>src</code>	Contains a handle to the source image. The source image must be stored in a pixel map structure. The compressor uses only the bit depth of this image to determine the compression time. You may set this parameter to <code>nil</code> if you are interested only in information about quality settings.
------------------	---

Image Compression Manager

<code>srcRect</code>	Contains a pointer to a rectangle defining the portion of the source image to compress. You may set this parameter to <code>nil</code> if you are interested only in information about quality settings. <code>GetCompressionTime</code> then uses the bounds of the source pixel map.								
<code>colorDepth</code>	Specifies the depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the <code>GetCodecInfo</code> function (see “Getting Information About Compressor Components” on page 3-66 for more information on the <code>GetCodecInfo</code> function).								
<code>cType</code>	Specifies a compressor type. You must set this parameter to a valid compressor type (see Table 3-3 on page 3-68 for a list of the available compressor types).								
<code>codec</code>	Contains a compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use one of the special identifiers: <table> <tr> <td><code>anyCodec</code></td><td>Choose the first compressor of the specified type</td></tr> <tr> <td><code>bestSpeedCodec</code></td><td>Choose the fastest compressor of the specified type</td></tr> <tr> <td><code>bestFidelityCodec</code></td><td>Choose the most accurate compressor of the specified type</td></tr> <tr> <td><code>bestCompressionCodec</code></td><td>Choose the compressor that produces the smallest resulting data</td></tr> </table> <p>You can also specify a component instance. This may be useful if you have previously set some parameter on a specific instance of a <code>codec</code> field and want to make sure that the specified instance is used for that operation.</p>	<code>anyCodec</code>	Choose the first compressor of the specified type	<code>bestSpeedCodec</code>	Choose the fastest compressor of the specified type	<code>bestFidelityCodec</code>	Choose the most accurate compressor of the specified type	<code>bestCompressionCodec</code>	Choose the compressor that produces the smallest resulting data
<code>anyCodec</code>	Choose the first compressor of the specified type								
<code>bestSpeedCodec</code>	Choose the fastest compressor of the specified type								
<code>bestFidelityCodec</code>	Choose the most accurate compressor of the specified type								
<code>bestCompressionCodec</code>	Choose the compressor that produces the smallest resulting data								
<code>spatialQuality</code>	Contains a pointer to a field containing the desired compressed image quality. The Image Compression Manager sets this field to the closest actual quality that the compressor can achieve. See “Compression Quality Constants” beginning on page 3-61 for valid values. If you are not interested in this information, pass <code>nil</code> in this parameter.								
<code>temporalQuality</code>	Contains a pointer to a field containing the desired temporal quality. Use this value only with images that are part of image sequences. The Image Compression Manager sets this field to the closest actual quality that the compressor can achieve. See “Compression Quality Constants” beginning on page 3-61 for valid values. If you are not interested in this information, pass <code>nil</code> in this parameter.								

Image Compression Manager

`compressTime`
Contains a pointer to a field to receive the compression time, in milliseconds. If the compressor cannot determine the amount of time required to compress the image or if the compressor does not support this function, this field is set to 0. If you are not interested in this information, pass `nil` in this parameter.

DESCRIPTION

The Image Compression Manager returns the maximum compression time for the specified image and parameters. Note that some compressors may not support this function. If the component you specify does not support this function, the Image Compression Manager returns a time value of 0.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor

GetSimilarity

The `GetSimilarity` function compares a compressed image to a picture stored in a pixel map and returns a value indicating the relative similarity of the two images.

```
pascal OSErr GetSimilarity (PixMapHandle src, const Rect *srcRect,
                             ImageDescriptionHandle desc, Ptr data,
                             Fixed *similarity);
```

<code>src</code>	Contains a handle to the noncompressed image. The image must be stored in a pixel map structure.
<code>srcRect</code>	Contains a pointer to a rectangle defining the portion of the image to compare to the compressed image. This rectangle should be the same size as the image described by the image description structure specified by the <code>desc</code> parameter.
<code>desc</code>	Specifies a handle to the image description structure that defines the compressed image for the operation.
<code>data</code>	Points to the compressed image data. This pointer must contain a 32-bit clean address.
<code>similarity</code>	Contains a pointer to a field that is to receive the similarity value. The compressor sets this field to reflect the relative similarity of the two images. Valid values range from 0 (completely different) to 1.0 (identical).

Image Compression Manager

RESULT CODES

noErr	0	No error
paramErr	-50	Invalid parameter specified
memFullErr	-108	Not enough memory available
noCodecErr	-8961	The Image Compression Manager could not find the specified compressor

GetCompressedImageSize

The `GetCompressedImageSize` function determines the size, in bytes, of a compressed image.

Most applications do not need to use this function because compressed images have a corresponding image description structure with a size field. You only need to use this function if you do not have an image description structure associated with your data—for example, when you are taking a compressed image out of a movie one frame at a time.

```
pascal OSErr GetCompressedImageSize (ImageDescriptionHandle desc,
                                     Ptr data, long bufferSize,
                                     DataProcRecordPtr dataProc,
                                     long *dataSize);
```

desc	Specifies a handle to the image description structure that defines the compressed image for the operation.
data	Points to the compressed image data. This pointer must contain a 32-bit clean address.
bufferSize	Specifies the size of the buffer to be used by the data-loading function specified by the <code>dataProc</code> parameter. If you have not specified a data-loading function, set this parameter to 0.
dataProc	Points to a data-loading function structure. If the data stream is not all in memory when your program calls <code>GetCompressedImageSize</code> , the compressor calls a function you provide that loads more compressed data (see “Data-Loading Functions” beginning on page 3-153 for more information about data-loading functions). If you have not provided a data-loading function, set this parameter to <code>nil</code> . In this case, the entire image must be in memory at the location specified by the <code>data</code> parameter.
dataSize	Contains a pointer to a field that is to receive the size, in bytes, of the compressed image.

DESCRIPTION

Your application may use the `GetCompressedImageSize` function when parsing a data stream that does not contain an image description structure for each frame in the sequence.

Image Compression Manager

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>noCodecErr</code>	-8961	Image Compression Manager could not find the specified compressor
<code>codecSpoolErr</code>	-8966	Error loading or unloading data
Component Manager errors		

Working With Images

This section discusses the functions that allow your application to compress and decompress single-frame images stored as pixel maps (of data type `PixelFormat`). See “Working With Sequences,” which begins on page 3-110, for information on compressing and decompressing sequences of images. See “Working With Pictures and PICT Files,” which begins on page 3-92, for information on compressing and manipulating single-frame images stored as pictures or picture files (in PICT format).

The Image Compression Manager provides two sets of functions for compressing and decompressing images. If you do not need to assert a lot of control over the compression operation, you can use the `CompressImage` and `DecompressImage` functions to work with compressed images. If you need more control over the compression parameters, you can use the `FCompressImage` and `FDecompressImage` functions.

You can convert a compressed image from one compression format to another by calling the `ConvertImage` function.

You can alter the spatial characteristics of a compressed image by calling the `TrimImage` function.

You can work with an image’s color table with the `SetImageDescriptionCTable` and `GetImageDescriptionCTable` functions.

CompressImage

The `CompressImage` function allows your application to compress a single-frame image that is currently stored as a pixel map structure.

```
pascal OSErr CompressImage (PixelFormatHandle src,
                             const Rect *srcRect,
                             CodecQ quality, CodecType cType,
                             ImageDescriptionHandle desc,
                             Ptr data);
```

<code>src</code>	Contains a handle to the image to be compressed. The image must be stored in a pixel map structure.
<code>srcRect</code>	Contains a pointer to a rectangle defining the portion of the image to compress.

Image Compression Manager

quality	Specifies the desired compressed image quality. See “Compression Quality Constants” on page 3-61 for valid values.
cType	Specifies a compressor type. You must set this parameter to a valid compressor type (see Table 3-3 on page 3-68 for a list of the available compressor types).
desc	Contains a handle that is to receive a formatted image description structure. The Image Compression Manager resizes this handle for the returned image description structure. Your application should store this image description with the compressed image data.
data	Points to a location to receive the compressed image data. It is your program’s responsibility to make sure that this location can receive at least as much data as indicated by the <code>GetMaxCompressionSize</code> function (described on page 3-72). The Image Compression Manager places the actual size of the compressed image into the <code>dataSize</code> field of the image description structure referred to by the <code>desc</code> parameter. This pointer must contain a 32-bit clean address. If you use a dereferenced, locked handle, you must call the Memory Manager’s <code>StripAddress</code> routine before you use that handle with this parameter (for details on <code>StripAddress</code> , see <i>Inside Macintosh: Memory</i>).

DESCRIPTION

The `CompressImage` function presents a simplified interface to your application, eliminating some parameters for the sake of convenience.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>memFullErr</code>	-108	Not enough memory available
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor

SEE ALSO

If you need to exert greater control over the compression operation, use the `FCompressImage` function, described in the next section.

FCompressImage

Like the CompressImage function, the FCompressImage function allows your application to compress a single-frame image that is currently stored as a pixel map structure (PixMap).

```
pascal OSErr FCompressImage (PixMapHandle src,
                             const Rect *srcRect,
                             short colorDepth, CodecQ quality,
                             CodecType cType,
                             CompressorComponent codec,
                             CTabHandle clut, CodecFlags flags,
                             long bufferSize,
                             FlushProcRecordPtr flushProc,
                             ProgressProcRecordPtr progressProc,
                             ImageDescriptionHandle desc,
                             Ptr data);
```

src	Contains a handle to the image to be compressed. The image must be stored in a pixel map structure.
srcRect	Contains a pointer to a rectangle defining the portion of the image to compress.
colorDepth	Specifies the depth at which the image is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the compressed image. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the GetCodecInfo function (see “Getting Information About Compressor Components” on page 3-66 for more information on the GetCodecInfo function).
quality	Specifies the desired compressed image quality. See “Compression Quality Constants” on page 3-61, for valid values.
cType	Specifies a compressor type. You must set this parameter to a valid compressor type (see Table 3-3 on page 3-68 for a list of the available compressor types).

Image Compression Manager

<code>codec</code>	<p>Specifies a compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use one of the special identifiers:</p> <p><code>anyCodec</code> Choose the first compressor of the specified type</p> <p><code>bestSpeedCodec</code> Choose the fastest compressor of the specified type</p> <p><code>bestFidelityCodec</code> Choose the most accurate compressor of the specified type</p> <p><code>bestCompressionCodec</code> Choose the compressor that produces the smallest resulting data</p> <p>You can also specify a component instance. This may be useful if you have previously set some parameter on a specific instance of a <code>codec</code> field and want to make sure that the specified instance is used for that operation.</p> <p>If you set the <code>codec</code> parameter to <code>anyCodec</code>, the Image Compression Manager chooses the first compressor it finds of the specified type.</p>
<code>clut</code>	<p>Contains a handle to a custom color lookup table. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the <code>colorDepth</code> parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the <code>ctSeed</code> field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the <code>colorDepth</code> parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the <code>clut</code> parameter when <code>colorDepth</code> is set to 33, 34, 36, or 40. If you set this parameter to <code>nil</code>, the compressor uses the color lookup table from the source pixel map.</p>
<code>flags</code>	<p>Specifies flags providing further control information. See “Image Compression Manager Function Control Flags,” which begins on page 3-62, for information about <code>CodecFlags</code> fields. The following flag is available for this function:</p> <p><code>codecFlagWasCompressed</code></p> <p>Indicates to the compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. Set this flag to 1 to indicate that the image was previously compressed. Set this flag to 0 if the image was not previously compressed.</p>

Image Compression Manager

<code>bufferSize</code>	Specifies the size of the buffer to be used by the data-unloading function specified by the <code>flushProc</code> parameter. If you have not specified a data-unloading function, set this parameter to 0.
<code>flushProc</code>	Points to a data-unloading function structure. If there is not enough memory to store the compressed image, the compressor calls a function you provide that unloads some of the compressed data (see “Data-Unloading Functions” beginning on page 3-154 for more information on the data-unloading structure). If you have not provided a data-unloading function, set this parameter to <code>nil</code> . In this case, the compressor writes the entire compressed image into the memory location specified by the <code>data</code> parameter.
<code>progressProc</code>	Points to a progress function structure. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress (see “Progress Functions” on page 3-156 for more information about progress functions). If you have not provided a progress function, set this parameter to <code>nil</code> . If you pass a value of <code>-1</code> , you obtain a standard progress function.
<code>desc</code>	Contains a handle that is to receive a formatted image description structure. The Image Compression Manager resizes this handle for the returned image description structure. Your application should store this image description with the compressed image data.
<code>data</code>	Points to a location to receive the compressed image data. It is your program’s responsibility to make sure that this location can receive at least as much data as indicated by the <code>GetMaxCompressionSize</code> function (described on page 3-72). If there is not sufficient memory to store the compressed image, you may choose to write the compressed data to mass storage during the compression operation. Use the <code>flushProc</code> parameter to identify your data-unloading function to the compressor. This pointer must contain a 32-bit clean address. If you use a dereferenced, locked handle, you must call the Memory Manager’s <code>StripAddress</code> routine before you use that handle with this parameter. (See <i>Inside Macintosh: Memory</i> for details on <code>StripAddress</code> .) The Image Compression Manager places the actual size of the compressed image into the <code>dataSize</code> field of the image description structure referred to by the <code>desc</code> parameter.

DESCRIPTION

The `FCompressImage` function gives your application additional control over the parameters that guide the compression operation.

Image Compression Manager

RESULT CODES

noErr	0	No error
paramErr	-50	Invalid parameter specified
memFullErr	-108	Not enough memory available
noCodecErr	-8961	The Image Compression Manager could not find the specified compressor
codecSpoolErr	-8966	Error loading or unloading data
codecAbortErr	-8967	Operation aborted by the progress function

SEE ALSO

If you find that you do not need this level of compression parameter control, use the `CompressImage` function, described in the previous section.

DecompressImage

The `DecompressImage` function allows your application to decompress a single-frame image into a pixel map structure. If you call this function when you have a picture open, the Image Compression Manager inserts the compressed image data into the picture.

```
pascal OSErr DecompressImage (Ptr data,
                               ImageDescriptionHandle desc,
                               PixMapHandle dst, const Rect *srcRect,
                               const Rect *dstRect, short mode,
                               RgnHandle mask);
```

data	Points to the compressed image data. This pointer must contain a 32-bit clean address. If you use a dereferenced, locked handle, you must call the Memory Manager's <code>StripAddress</code> routine before you use that handle with this parameter.
desc	Contains a handle to the image description structure that describes the compressed image.
dst	Contains a handle to the pixel map where the decompressed image is to be displayed. Set the current graphics port to the port that contains this pixel map.
srcRect	Contains a pointer to a rectangle defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by (0,0) and <code>((**desc).width, (**desc).height)</code> . If you want to decompress the entire source image, set this parameter to <code>nil</code> . If the parameter is <code>nil</code> , the rectangle is set to the rectangle structure of the image description structure.

Image Compression Manager

<code>dstRect</code>	Contains a pointer to the rectangle into which the decompressed image is to be loaded. The compressor scales the source image to fit into this destination rectangle.
<code>mode</code>	Specifies the transfer mode for the operation. The Image Compression Manager supports the same transfer modes supported by QuickDraw's <code>CopyBits</code> routine.
<code>mask</code>	Contains a handle to a clipping region in the destination coordinate system. If specified, the compressor applies this mask to the destination image. If you do not want to mask bits in the destination, set this parameter to <code>nil</code> .

DESCRIPTION

The `DecompressImage` function presents a simplified interface to your application, eliminating some parameters for the sake of convenience.

Note that the `DecompressImage` function is invoked through the `StdPix` function (see “Working With the `StdPix` Function,” which begins on page 3-141, for more information).

SPECIAL CONSIDERATIONS

The graphics port and the graphics device should be set to the destination before you call the `DecompressImage` function.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>memFullErr</code>	-108	Not enough memory available
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor

SEE ALSO

If you need to exert greater control over the decompression operation, use the `FDecompressImage` function, described in the next section.

FDecompressImage

Like the `DecompressImage` function, the `FDecompressImage` function allows your application to decompress a single-frame image into a pixel map (`PixelFormat`). This function gives your application greater control over the parameters that guide the

Image Compression Manager

decompression operation. If you find that you do not need this level of control, use the `DecompressImage` function, described in the previous section.

```
pascal OSErr FDecompressImage (Ptr data,
                                ImageDescriptionHandle desc,
                                PixMapHandle dst,
                                const Rect *srcRect,
                                MatrixRecordPtr matrix,
                                short mode, RgnHandle mask,
                                PixMapHandle matte,
                                const Rect *matteRect,
                                CodecQ accuracy,
                                DecompressorComponent codec,
                                long bufferSize,
                                DataProcRecordPtr dataProc,
                                ProgressProcRecordPtr progressProc);
```

<code>data</code>	Points to the compressed image data. If the entire compressed image cannot be stored at this location, your application may provide a data-loading function (see the discussion of the <code>dataProc</code> parameter to this function). This pointer must contain a 32-bit clean address. If you use a dereferenced, locked handle, you must call the Memory Manager's <code>StripAddress</code> routine before you use that handle with this parameter. (See <i>Inside Macintosh: Memory</i> for details on <code>StripAddress</code> .)
<code>desc</code>	Contains a handle to the image description structure that describes the compressed image.
<code>dst</code>	Contains a handle to the pixel map where the decompressed image is to be displayed. Set the current graphics port to the port that contains this pixel map.
<code>srcRect</code>	Contains a pointer to a rectangle defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by (0,0) and <code>((**desc).width, (**desc).height)</code> . If you want to decompress the entire source image, set this parameter to <code>nil</code> . If the parameter is <code>nil</code> , the rectangle is set to the rectangle structure of the image description structure.
<code>matrix</code>	Points to a matrix structure that specifies how to transform the image during decompression. You can use the matrix structure to translate or scale the image during decompression. If you do not want to apply such effects, set the <code>matrix</code> parameter to <code>nil</code> . See the chapter "Movie Toolbox" in this book for more information about matrix operations.
<code>mode</code>	Specifies the transfer mode for the operation. The Image Compression Manager supports the same transfer modes supported by QuickDraw's <code>CopyBits</code> routine (described in <i>Inside Macintosh: Imaging</i>).

Image Compression Manager

mask	Contains a handle to a clipping region in the destination coordinate system. If specified, the decompressor applies this mask to the destination image. If you do not want to mask bits in the destination, set this parameter to <code>nil</code> .
matte	Contains a handle to a pixel map that contains a blend matte. You can use the blend matte to cause the decompressed image to be blended into the destination pixel map. The matte can be defined at any supported pixel depth—the matte depth need not correspond to the source or destination depths. However, the matte must be in the coordinate system of the source image. If you do not want to apply a blend matte, set this parameter to <code>nil</code> .
matteRect	Contains a pointer to a rectangle defining a portion of the blend matte to apply. If you do not want to use the entire matte referred to by the <code>matte</code> parameter, use this parameter to specify a rectangle within that matte. If specified, this rectangle must be the same size as the rectangle specified by the <code>srcRect</code> parameter. If you want to use the entire matte, or if you are not providing a blend matte, set this parameter to <code>nil</code> .
accuracy	Specifies the accuracy desired in the decompressed image. Values for this parameter are on the same scale as compression quality. See “Compression Quality Constants” beginning on page 3-61 for valid values. (For a good display of still images, you should specify at least the <code>codecHighQuality</code> constant.)
codec	<p>Contains a compressor identifier. Specify a particular decompressor by setting this parameter to its compressor identifier. Alternatively, you may use one of the special identifiers:</p> <p><code>anyCodec</code> Choose the first decompressor of the specified type</p> <p><code>bestSpeedCodec</code> Choose the fastest decompressor of the specified type</p> <p><code>bestFidelityCodec</code> Choose the most accurate decompressor of the specified type</p> <p><code>bestCompressionCodec</code> Choose the decompressor that produces the smallest resulting data</p> <p>You can also specify a component instance. This may be useful if you have previously set some parameter on a specific instance of a <code>codec</code> field and want to make sure that the specified instance is used for that operation.</p> <p>If you set the <code>codec</code> parameter to <code>anyCodec</code>, the Image Compression Manager chooses the first decompressor it finds of the specified type.</p>
bufferSize	Specifies the size of the buffer to be used by the data-loading function specified by the <code>dataProc</code> parameter. If you have not specified a data-loading function, set this parameter to 0.

Image Compression Manager

dataProc Points to a data-loading function structure. If there is not enough memory to store the compressed image, the compressor calls a function you provide that loads more compressed data (see “Data-Loading Functions” on page 3-153 for more information about data-loading functions). If you have not provided a data-loading function, set this parameter to `nil`. In this case, the compressor expects that the entire compressed image is in the memory location specified by the `data` parameter.

progressProc Points to a progress function structure. During the decompression operation, the compressor may occasionally call a function you provide in order to report its progress (see “Progress Functions” on page 3-156 for more information about progress functions). If you have not provided a progress function, set this parameter to `nil`. If you pass a value of `-1`, you obtain a standard progress function.

DESCRIPTION

Note that this function is invoked through the `StdPix` function (see “Working With the `StdPix` Function,” which begins on page 3-141, for more information).

SPECIAL CONSIDERATIONS

The graphics port and the graphics device should be set to the destination before you call the `FDecompressImage` function.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>memFullErr</code>	-108	Not enough memory available
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor
<code>codecSpoolErr</code>	-8966	Error loading or unloading data
<code>codecAbortErr</code>	-8967	Operation aborted by the progress function

ConvertImage

The `ConvertImage` function allows your application to convert the format of a compressed image. This function is essentially equivalent to decompressing and recompressing the image.

Image Compression Manager

```
pascal OSErr ConvertImage (ImageDescriptionHandle srcDD,
                           Ptr srcData, short colorDepth,
                           CTabHandle clut, CodecQ accuracy,
                           CodecQ quality, CodecType cType,
                           CodecComponent codec,
                           ImageDescriptionHandle dstDD,
                           Ptr dstData);
```

srcDD	Contains a handle to the image description structure that describes the compressed image.
srcData	Points to the compressed image data. This pointer must contain a 32-bit clean address.
colorDepth	Specifies the depth at which the recompressed image is likely to be viewed. Decompressors may use this as an indication of the color or grayscale resolution of the compressed image. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the <code>GetCodecInfo</code> function (see “Getting Information About Compressor Components” on page 3-66 for more information on the <code>GetCodecInfo</code> function).
clut	Contains a handle to a custom color lookup table. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the <code>colorDepth</code> parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the <code>ctSeed</code> field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the <code>colorDepth</code> parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the <code>clut</code> parameter when <code>colorDepth</code> is set to 33, 34, 36, or 40. If you set this parameter to <code>nil</code> , the compressor uses the color lookup table from the source image description structure.
accuracy	Specifies the accuracy desired in the decompressed image. Values for this parameter are on the same scale as compression quality. See “Compression Quality Constants” on page 3-61 for valid values. (For a good display of still images, you should specify at least the <code>codecHighQuality</code> constant.)

Image Compression Manager

quality	Specifies the desired compressed image quality. See “Compression Quality Constants” on page 3-61 for valid values. Use the following value: codecHighQuality Specifies high-quality image reproduction. This value should correspond to the highest image quality that can be achieved with reasonable performance.
cType	Specifies a compressor type. You must set this parameter to a valid compressor type. See Table 3-3 on page 3-68 for a list of the available compressor types.
codec	Contains a compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use one of the special identifiers: anyCodec Choose the first compressor of the specified type bestSpeedCodec Choose the fastest compressor of the specified type bestFidelityCodec Choose the most accurate compressor of the specified type You can also specify a component instance. This may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation. If you set the codec parameter to anyCodec, the Image Compression Manager chooses the first compressor it finds of the specified type.
dstDD	Contains a handle that is to receive a formatted image description structure. The Image Compression Manager resizes this handle for the returned image description structure. Your application should store this image description with the compressed image data.
dstData	Points to a location to receive the compressed image data. It is your program’s responsibility to make sure that this location can receive at least as much data as indicated by the GetMaxCompressionSize function (described on page 3-72). The Image Compression Manager places the actual size of the compressed image into the dataSize field of the image description referred to by the dstDD parameter. This pointer must contain a 32-bit clean address.

DESCRIPTION

During the decompression operation, the decompressor uses the srcDD, srcData, and accuracy parameters. During the subsequent compression operation, the compressor uses the colorDepth, clut, cType, codec, quality, dstDD, and dstData parameters.

RESULT CODES

noErr	0	No error
paramErr	-50	Invalid parameter specified
memFullErr	-108	Not enough memory available
noCodecErr	-8961	The Image Compression Manager could not find the specified compressor

TrimImage

The TrimImage function adjusts a compressed image to the boundaries defined by a rectangle specified by your application.

```
pascal OSErr TrimImage (ImageDescriptionHandle desc, Ptr inData,
                        long inBufferSize,
                        DataProcRecordPtr dataProc,
                        Ptr outData, long outBufferSize,
                        FlushProcRecordPtr flushProc,
                        Rect *trimRect,
                        ProgressProcRecordPtr progressProc);
```

desc	Contains a handle to the image description structure that describes the compressed image. On return from TrimImage, the compressor updates this image description to refer to the resized image.
inData	Points to the compressed image data. If the entire compressed image cannot be stored at this location, your application may provide a data-loading function (see the discussion of the dataProc parameter to this function). This pointer must contain a 32-bit clean address.
inBufferSize	Specifies the size of the buffer to be used by the data-loading function specified by the dataProc parameter. If you have not specified a data-loading function, this parameter is ignored.
dataProc	Points to a data-loading function structure. If there is not enough memory to store the compressed image, the compressor calls a function you provide that loads more compressed data (see “Data-Loading Functions” beginning on page 3-153 for more information about data-loading function structures). If you have not provided a data-loading function, set this parameter to nil. In this case, the compressor expects that the entire compressed image is in the memory location specified by the inData parameter.
outData	Points to a buffer to receive the trimmed image. Your application should create this destination buffer at least as large as the source image. If there is not sufficient memory to store the compressed image, you may choose to write the compressed data to mass storage during the compression operation. Use the flushProc parameter to identify your data-unloading function to the compressor. This pointer must contain a 32-bit clean address.

Image Compression Manager

The Image Compression Manager places the actual size of the resulting image into the `dataSize` field of the image description structure referred to by the `desc` parameter.

<code>outBufferSize</code>	Specifies the size of the buffer to be used by the data-unloading function specified by the <code>flushProc</code> parameter. If you have not specified a data-unloading function, this parameter is ignored.
<code>flushProc</code>	Points to a data-unloading function structure. If there is not enough memory to store the compressed image, the compressor calls a function you provide that unloads some of the compressed data (see “Data-Unloading Functions” beginning on page 3-154 for more information on the data-unloading structure). If you have not provided a data-unloading function, set this parameter to <code>nil</code> . In this case, the compressor writes the entire compressed image into the memory location specified by the <code>data</code> parameter.
<code>trimRect</code>	Contains a pointer to a rectangle that defines the desired image dimensions. Upon return to your application, the compressor adjusts the rectangle values so that they refer to the same rectangle in the result image (this is necessary whenever data is removed from the beginning or from the left side of the image).
<code>progressProc</code>	Points to a progress function structure. During the operation, the compressor may occasionally call a function you provide in order to report its progress (see “Progress Functions” on page 3-156 for more information about progress functions). If you have not provided a progress function, set this parameter to <code>nil</code> . If you pass a value of <code>-1</code> , you obtain a standard progress function.

DESCRIPTION

The resulting image data is still compressed and is in the same compression format as the source image.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>memFullErr</code>	-108	Not enough memory available
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor
<code>codecUnimpErr</code>	-8962	Feature not implemented by this compressor
<code>codecSpoolErr</code>	-8966	Error loading or unloading data
<code>codecAbortErr</code>	-8967	Operation aborted by the progress function

SetImageDescriptionCTable

Your application may use the `SetImageDescriptionCTable` function to update the custom color table for an image. The Image Compression Manager copies the custom color table for an image into the appropriate image description structure. This function does not change the image data, just the color table.

<code>pascal OSErr SetImageDescriptionCTable</code>		
<code>(ImageDescriptionHandle desc,</code>		
<code>CTabHandle ctable);</code>		
<code>desc</code>		Contains a handle to the appropriate image description structure. The <code>SetImageDescriptionCTable</code> function updates the size of the image description to accommodate the new color table and removes the old color table, if one is present.
<code>ctable</code>		Contains a handle to the new color table. The <code>SetImageDescriptionCTable</code> function loads this color table into the image description referred to by the <code>desc</code> parameter. Set this parameter to <code>nil</code> to remove a color table.

DESCRIPTION

The `SetImageDescriptionCTable` function is rarely used. Typically, you supply the color table when your application compresses an image. The Image Compression Manager stores the color table with the image.

RESULT CODES

<code>noErr</code>	<code>0</code>	No error
<code>paramErr</code>	<code>-50</code>	Invalid parameter specified
<code>memFullErr</code>	<code>-108</code>	Not enough memory available
<code>noCodecErr</code>	<code>-8961</code>	The Image Compression Manager could not find the specified compressor

GetImageDescriptionCTable

Your application may use the `GetImageDescriptionCTable` function to set the custom color table for an image.

<code>pascal OSErr GetImageDescriptionCTable</code>		
<code>(ImageDescriptionHandle desc,</code>		
<code>CTabHandle *ctable);</code>		
<code>desc</code>		Contains a handle to the appropriate image description structure.

Image Compression Manager

`ctable` Contains a pointer to a field that is to receive a color table handle. The `GetImageDescriptionCTable` function returns the color table for the image described by the image description structure that is referred to by the `desc` parameter. The function correctly sizes the handle for the color table it returns.

DESCRIPTION

The Image Compression Manager stores the custom color table for an image in the appropriate image description structure. Your application must use QuickDraw's `DisposeCTable` routine to free the color table. (For details on `DisposeCTable`, see *Inside Macintosh: Imaging*.)

SPECIAL CONSIDERATIONS

If you want to find out if there is a custom color table, you should check the size of the `CTSize` or `CTSeed` fields in the returned `ctable` parameter. If `CTSize` is 0 or if the `CTSeed` field is less than 0, then the color table is not a custom color table for that image.

RESULT CODES

<code>noErr</code>	0	No error
<code>paramErr</code>	-50	Invalid parameter specified
<code>memFullErr</code>	-108	Not enough memory available
<code>noCodecErr</code>	-8961	The Image Compression Manager could not find the specified compressor

Working With Pictures and PICT Files

This section describes the functions that let your application compress and decompress single-frame images stored as pictures and PICT files. See "Working With Images," which begins on page 3-77, for information on compressing and manipulating single-frame images stored as pixel map structures. See "Working With Sequences," which begins on page 3-110, for information on compressing and decompressing sequences of images.

As with image compression, the Image Compression Manager provides two sets of functions for working with compressed pictures. If you do not need to control the compression parameters, use the `CompressPicture` or `CompressPictureFile` functions. If you need more control over the operation, use the `FCompressPicture` or `FCompressPictureFile` functions.

The Image Compression Manager automatically expands compressed pictures when you display them. Use the `DrawPictureFile` function to display the contents of a picture file. If you want to alter the spatial characteristics of the image, use the `DrawTrimmedPicture` or `DrawTrimmedPictureFile` functions.

You can work with an image's control information by calling the `GetPictureFileHeader` function.