Compression and Decompression Reference for QuickTime

QuickTime > Compression & Decompression

2006-05-23

Contents

Compression and Decompression Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	IOMacOSTypes.h ImageCompression.h MacTypes.h OSUtils.h

Overview

QuickTime compression and decompression APIs help applications compress and decompress movie data.

Functions by Task

Aligning Windows

`AlignScreenRect` (page 21)
: Aligns a specified rectangle to the strictest screen that the rectangle intersects.

`AlignWindow` (page 22)
: Moves a specified window to the nearest optimal alignment position.

`DragAlignedGrayRgn` (page 63)
: Drags a specified gray region along an optimal alignment grid.

`DragAlignedWindow` (page 64)
: Drags the specified window along an optimal alignment grid.

Applying Matrix Transformations

`TransformFixedPoints` (page 168)
: Transforms a set of fixed points through a specified matrix.

`TransformFixedRect` (page 168)
: Transforms the upper-left and lower-right points of a rectangle through a matrix that is specified by fixed points.

`TransformPoints` (page 169)
: Transforms a set of QuickDraw points through a specified matrix.

`TransformRect` (page 170)
: Transforms the upper-left and lower-right points of a rectangle through a specified matrix.

`TransformRgn` (page 171)
: Applies a specified matrix to a region.

Changing Sequence-Compression Parameters

`GetCSequenceMaxCompressionSize` (page 92)
: Determines the maximum size an image will be after compression for a given compression sequence.

`GetCSequencePrevBuffer` (page 93)
: Determines the location of the previous image buffer allocated by the compressor.

`SetCSequenceFlushProc` (page 150)
: Assigns a data-unloading function to a sequence.

`SetCSequenceKeyFrameRate` (page 152)
: Adjusts the key frame rate for the current sequence.

`SetCSequencePreferredPacketSize` (page 152)
: Sets the preferred packet size for a sequence.

`SetCSequencePrev` (page 153)
: Allows the application to set the pixel map and boundary rectangle used by the previous frame in temporal compression.

`SetCSequenceQuality` (page 154)
: Adjusts the spatial or temporal quality for the current sequence.

Changing Sequence-Decompression Parameters

`GetDSequenceImageBuffer` (page 94)
: Determines the location of the offscreen image buffer allocated by a decompressor.

`GetDSequenceScreenBuffer` (page 96)
: Determines the location of the offscreen screen buffer allocated by a decompressor.

`PtInDSequenceData` (page 129)
: Tests to see if a compressed image contains data at a a given point.

`SetDSequenceAccuracy` (page 155)
: Adjusts the decompression accuracy for the current sequence.

`SetDSequenceDataProc` (page 156)
: Assigns a data-loading function to a sequence.

`SetDSequenceMask` (page 158)
: Assigns a clipping region to a sequence.

`SetDSequenceMatrix` (page 158)
: Assigns a mapping matrix to a sequence.

`SetDSequenceMatte` (page 159)
: Assigns a blend matte to a sequence.

`SetDSequenceSrcRect` (page 161)
: Defines the portion of an image to decompress.

`SetDSequenceTimeCode` (page 162)
: Sets the timecode value for a frame that is about to be decompressed.

`SetDSequenceTransferMode` (page 162)
: Sets the mode used when drawing a decompressed image.

Constraining Compressed Data

`GetCSequenceDataRateParams` (page 91)
: Obtains the data rate parameters previously set with SetCSequenceDataRateParams.

`SetCSequenceDataRateParams` (page 149)
: Communicates information to compressors that can constrain compressed data in a particular sequence to a specific data rate.

Controlling Hardware Scaling

`GDGetScale` (page 83)
: Returns the current scale of the given screen graphics device.

`GDHasScale` (page 83)
: Returns the closest possible scaling that a particular screen device can be set to in a given pixel depth.

`GDSetScale` (page 84)
: Sets a screen graphics device to a new scale.

Creating an Effect Sample Description

`MakeImageDescriptionForEffect` (page 123)
: Returns an ImageDescription structure you can use to help create a sample description for an effect.

Creating File Previews

`AddFilePreview` (page 20)
: Adds a preview to a file.

`MakeFilePreview` (page 122)
: Creates a preview for a file.

Getting Information About Compressed Data

`GetCompressedImageSize` (page 87)
: Determines the size, in bytes, of a compressed image.

`GetCompressionTime` (page 89)
: Determines the estimated amount of time required to compress a given image.

`GetMaxCompressionSize` (page 104)
: Determines the maximum size an image will be after compression.

`GetSimilarity` (page 107)
: Compares a compressed image to a picture stored in a pixel map and returns a value indicating the relative similarity of the two images.

Getting Information About Compressor Components

`CodecManagerVersion` (page 34)
Determines the version of the installed Image Compression Manager.

`DisposeCodecNameList` (page 62)
Disposes of the compressor name list structure you obtained by calling GetCodecNameList.

`FindCodec` (page 79)
Determines which of the installed compressors or decompressors has been chosen to field requests made by using one of the special compressor identifiers.

`GetCodecInfo` (page 86)
Returns information about a single compressor component.

`GetCodecNameList` (page 86)
Retrieves a list of installed compressor components or types.

Image Compression Manager Utility Functions

`ICMDecompressComplete` (page 109)
Signals the completion of a decompression operation.

`ICMShieldSequenceCursor` (page 115)
Hides the cursor during decompression operations.

Image Transcoder Support

`ImageTranscodeDisposeFrameData` (page 119)
Disposes transcoded image data.

`ImageTranscodeFrame` (page 119)
Transcodes a frame of image data.

`ImageTranscodeSequenceBegin` (page 120)
Initiates an image transcoder sequence operation.

`ImageTranscodeSequenceEnd` (page 121)
Ends an image transcoder sequence operation.

Making Thumbnail Pictures

`MakeThumbnailFromPicture` (page 125)
Creates a thumbnail picture from a specified Picture structure.

`MakeThumbnailFromPictureFile` (page 126)
Creates a thumbnail picture from a specified picture file.

`MakeThumbnailFromPixMap` (page 127)
Creates a thumbnail picture from a specified PixMap structure.

Managing Matrices

`ConcatMatrix` (page 50)
: Concatenates two matrices, combining the transformations described by both matrices into a single matrix.

`CopyMatrix` (page 52)
: Copies the contents of one matrix into another matrix.

`EqualMatrix` (page 70)
: Compares two matrices and returns a result that indicates whether the matrices are equal.

`GetMatrixType` (page 103)
: Obtains information about a matrix.

`InverseMatrix` (page 121)
: Creates a new matrix that is the inverse of a specified matrix.

`MapMatrix` (page 128)
: Alters an existing matrix so that it defines a transformation from one rectangle to another.

`RectMatrix` (page 144)
: Creates a matrix that performs the translate and scale operation described by the relationship between two rectangles.

`RotateMatrix` (page 146)
: Modifies the contents of a matrix so that it defines a rotation operation.

`ScaleMatrix` (page 147)
: Modifies the contents of a matrix so that it defines a scaling operation.

`SetIdentityMatrix` (page 163)
: Sets the contents of a matrix so that it performs no transformation.

`SkewMatrix` (page 165)
: Modifies the contents of a matrix so that it defines a skew transformation.

`TranslateMatrix` (page 171)
: Adds a translation value to a specified matrix.

Obtaining a Graphics Importer Instance

`GetGraphicsImporterForDataRef` (page 97)
: Locates and opens a graphics importer component that can be used to draw the image from specified data reference.

`GetGraphicsImporterForDataRefWithFlags` (page 98)
: Locates and opens a graphics importer component for a data reference with flags that control the search process.

`GetGraphicsImporterForFile` (page 99)
: Locates and opens a graphics importer component that can be used to draw a specified file.

Working With Graphics Devices and Graphics Worlds

`GetBestDeviceRect` (page 85)
: Selects the deepest of all available graphics devices, while treating 16-bit and 32-bit screens as having equal depth.

`NewImageGWorld` (page 128)
: Creates an offscreen graphics world.

Working With Image Descriptions

`AddImageDescriptionExtension` (page 20)
: Adds an extension to an ImageDescription structure.

`CountImageDescriptionExtensionType` (page 52)
: Counts the number of extensions of a given type in an ImageDescriptionHandle.

`GetImageDescriptionExtension` (page 102)
: Returns a new handle with the data from a specified image description extension.

`GetNextImageDescriptionExtensionType` (page 106)
: Retrieves an image description structure extension type.

`QTGetPixelFormatDepthForImageDescription` (page 130)
: For a given pixel format, returns the depth value that should be used in image descriptions.

`RemoveImageDescriptionExtension` (page 145)
: Removes a specified extension from an ImageDescription structure.

Working With Pictures and PICT Files

`CompressPicture` (page 41)
: Compresses a single-frame image stored as a picture structure and places the result in another picture.

`CompressPictureFile` (page 43)
: Compresses a single-frame image stored as a picture file and places the result in another picture file.

`DrawPictureFile` (page 67)
: Draws an image from a specified picture file in the current graphics port.

`DrawTrimmedPicture` (page 68)
: Draws an image that is stored as a picture into the current graphics port and trims that image to fit a specified region.

`DrawTrimmedPictureFile` (page 69)
: Draws an image that is stored as a picture file into the current graphics port and trims that image to fit a specified region.

`FCompressPicture` (page 73)
: Compresses a single-frame image stored as a picture structure and places the result in another picture, with added control over the compression process.

`FCompressPictureFile` (page 75)
: Compresses a single-frame image stored as a picture file and places the result in another picture file, with added control over the compression process.

`GetPictureFileHeader` (page 107)
: Extracts the picture frame and file header from a specified picture file.

Working With Pixel Maps

`CompressImage` (page 39)
: Compresses a single-frame image that is currently stored as a pixel map structure.

`ConvertImage` (page 50)
: Converts the format of a compressed image.

`DecompressImage` (page 53)
: Decompresses a single-frame image into a pixel map structure.

`FCompressImage` (page 71)
: Compresses a single-frame image that is currently stored as a pixel map structure, with added control over the compression process.

`FDecompressImage` (page 77)
: Decompresses a single-frame image into a pixel map structure, with added control over the decompression process.

`GetImageDescriptionCTable` (page 101)
: Gets the custom color table for an image.

`SetImageDescriptionCTable` (page 164)
: Updates the custom ColorTable structure for an image.

`TrimImage` (page 172)
: Adjusts a compressed image to the boundaries defined by a specified rectangle.

Working With Sequences

`CDSequenceBusy` (page 23)
: Checks the status of an asynchronous compression or decompression operation.

`CDSequenceChangedSourceData` (page 23)
: Notifies the compressor that the image source data has changed.

`CDSequenceDisposeDataSource` (page 24)
: Disposes of a data source.

`CDSequenceDisposeMemory` (page 24)
: Disposes of memory allocated by the codec.

`CDSequenceEnd` (page 25)
: Indicates the end of processing for an image sequence.

`CDSequenceEquivalentImageDescription` (page 25)
: Reports whether two image descriptions are the same.

`CDSequenceFlush` (page 27)
: Stops a decompression sequence, aborting processing of any queued frames.

`CDSequenceInvalidate` (page 28)
: Notifies the Image Compression Manager that the destination port for the given image decompression sequence has been invalidated.

`CDSequenceNewDataSource` (page 29)
Creates a new data source.

`CDSequenceNewMemory` (page 31)
Requests codec-allocated memory.

`CDSequenceSetSourceData` (page 32)
Sets data in a new frame to a specific data source.

`CompressSequenceBegin` (page 43)
Signals the beginning of the process of compressing a sequence of frames.

`CompressSequenceFrame` (page 46)
Compresses one of a sequence of frames.

`DecompressSequenceBegin` (page 55)
Obsolete. See DecompressSequenceBeginS.

`DecompressSequenceBeginS` (page 56)
Sends a sample image to a decompressor.

`DecompressSequenceFrame` (page 58)
Obsolete. See DecompressSequenceFrameS.

`DecompressSequenceFrameS` (page 59)
Queues a frame for decompression and specifies the size of the compressed data; new applications should use DecompressSequenceFrameWhen.

`DecompressSequenceFrameWhen` (page 60)
Queues a frame for decompression and specifies the time at which decompression will begin.

`SetSequenceProgressProc` (page 165)
Installs a progress procedure for a sequence.

Working With the StdPix Function

`GetCompressedPixMapInfo` (page 88)
Retrieves information about a compressed image.

`SetCompressedPixMapInfo` (page 148)
Stores information about a compressed image for StdPix.

`StdPix` (page 166)
Extends the grafProcs field of the CGrafPort structure to support compressed data, mattes, matrices, and pixel maps, letting you intercept image data in compressed form before it is decompressed and displayed.

Working With Video Fields

`ImageFieldSequenceBegin` (page 116)
Initiates an image field sequence operation and specifies the input and output data format.

`ImageFieldSequenceEnd` (page 117)
Ends an image field sequence operation.

`ImageFieldSequenceExtractCombine` (page 117)
Performs field operations on video data.

Supporting Functions

`CDSequenceEquivalentImageDescriptionS` (page 26)
Undocumented

`CDSequenceGetDataSource` (page 28)
Gets a data source for a decompression sequence.

`CDSequenceSetSourceDataQueue` (page 33)
Sets a data queue as the source for a decompression sequence.

`CDSequenceSetTimeBase` (page 33)
Sets a time base for a decompression sequence.

`CompAdd` (page 35)
Undocumented

`CompCompare` (page 35)
Undocumented

`CompDiv` (page 36)
Undocumented

`CompFixMul` (page 37)
Undocumented

`CompMul` (page 37)
Undocumented

`CompMulDiv` (page 38)
Undocumented

`CompMulDivTrunc` (page 38)
Undocumented

`CompNeg` (page 39)
Undocumented

`CompShift` (page 48)
Undocumented

`CompSquareRoot` (page 49)
Undocumented

`CompSub` (page 49)
Undocumented

`FixExp2` (page 80)
Undocumented

`FixLog2` (page 81)
Undocumented

`FixMulDiv` (page 81)
Undocumented

`FixPow` (page 82)
Undocumented

`FracSinCos` (page 82)
Undocumented

`GetCSequenceFrameNumber` (page 91)
Returns the current frame number of the specified sequence.

`GetCSequenceKeyFrameRate` (page 92)
: Determines the current key frame rate of a sequence.

`GetDSequenceMatrix` (page 95)
: Gets the matrix that was specified for a decompression sequence by a call to SetDSequenceMatrix, or that was set at DecompressSequenceBegin.

`GetDSequenceNonScheduledDisplayDirection` (page 95)
: Returns the display direction for a decompress sequence.

`GetDSequenceNonScheduledDisplayTime` (page 96)
: Gets the display time for a decompression sequence.

`GetGraphicsImporterForFileWithFlags` (page 100)
: Locates and opens a graphics importer component for a file with flags that control the search process.

`HitTestDSequenceData` (page 108)
: Undocumented

`ICMDecompressCompleteS` (page 110)
: Undocumented

`ICMGetPixelFormatInfo` (page 111)
: Retrieves pixel format information.

`ICMSequenceGetChainMember` (page 111)
: Undocumented

`ICMSequenceGetInfo` (page 112)
: Gets multiprocessing properties for compression and decompression sequences.

`ICMSequenceLockBits` (page 113)
: Undocumented

`ICMSequenceSetInfo` (page 113)
: Sets multiprocessing properties for compression and decompression sequences.

`ICMSequenceUnlockBits` (page 114)
: Undocumented

`ICMSetPixelFormatInfo` (page 115)
: Lets you define your own pixel format.

`MakeImageDescriptionForPixMap` (page 124)
: Fills out an ImageDescription structure corresponding to a PixMap structure.

`QTGetFileNameExtension` (page 130)
: Gets the extension to a file name.

`QTGetPixelSize` (page 131)
: Returns the bits per pixel for a given pixel format.

`QTGetPixMapHandleGammaLevel` (page 132)
: Retrieves the current PixMap extension's gamma level setting.

`QTGetPixMapHandleRequestedGammaLevel` (page 132)
: Retrieves the current PixMap extension's requested gamma level setting.

`QTGetPixMapHandleRowBytes` (page 133)
: Gets the rowBytes value for a pixel map accessed by a handle.

`QTGetPixMapPtrGammaLevel` (page 133)
: Retrieves the current PixMap extension's gamma level setting.

`QTGetPixMapPtrRequestedGammaLevel` (page 134)
: Retrieves the current PixMap extension's gamma level setting.

`QTGetPixMapPtrRowBytes` (page 134)
: Gets the rowBytes value for a pixel map accessed by a pointer.

`QTNewGWorld` (page 135)
: Creates an offscreen graphics world that may have a non-Macintosh pixel format.

`QTNewGWorldFromPtr` (page 137)
: Wraps a graphics world and pixel map structure around an existing block of memory containing an image.

`QTSetPixMapHandleGammaLevel` (page 138)
: Sets the gamma level of a pixel map.

`QTSetPixMapHandleRequestedGammaLevel` (page 139)
: Sets the requested gamma level of a pixel map.

`QTSetPixMapHandleRowBytes` (page 140)
: Sets the rowBytes value for a pixel map accessed by a handle.

`QTSetPixMapPtrGammaLevel` (page 140)
: Sets the gamma level of a pixel map.

`QTSetPixMapPtrRequestedGammaLevel` (page 141)
: Sets the requested gamma level of a pixel map.

`QTSetPixMapPtrRowBytes` (page 142)
: Sets the rowBytes value for a pixel map accessed by a pointer.

`QTUpdateGWorld` (page 142)
: Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world with a non-Macintosh pixel format.

`QuadToQuadMatrix` (page 144)
: Defines a matrix that maps between four input points and four output points.

`ReplaceDSequenceImageDescription` (page 146)
: Undocumented

`SetCSequenceFrameNumber` (page 151)
: Informs the compressor in use for the specified sequence that frames are being compressed out of order.

`SetDSequenceFlags` (page 157)
: Sets data loading flags.

`SetDSequenceNonScheduledDisplayDirection` (page 160)
: Sets the display direction for a decompress sequence.

`SetDSequenceNonScheduledDisplayTime` (page 160)
: Sets the display time for a decompression sequence.

`UnsignedFixMulDiv` (page 173)
: Performs multiplications and divisions on unsigned fixed-point numbers.

Functions

AddFilePreview

Adds a preview to a file.

```
OSErr AddFilePreview (
   short resRefNum,
   OSType previewType,
   Handle previewData
);
```

Parameters

resRefNum

The resource file for this operation. You must have opened this resource file with write permission. If there is a preview in the specified file, the Movie Toolbox replaces that preview with a new one.

previewType

The resource type to be assigned to the preview. This type should correspond to the type of data stored in the preview. For example, if you have created a QuickDraw picture that you want to use as a preview for a file, you should set the `previewType` parameter to `'PICT'`.

previewData

A handle to the preview data. For example, if the preview data is a picture, you would provide a picture handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You must have created the preview data yourself. If the specified file already has a preview defined, the `AddFilePreview` function replaces it with the new preview.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTMusicToo

Declared In

`ImageCompression.h`

AddImageDescriptionExtension

Adds an extension to an ImageDescription structure.

```
OSErr AddImageDescriptionExtension (
   ImageDescriptionHandle desc,
   Handle extension,
   long idType
);
```

Parameters

desc

A handle to the `ImageDescription` structure to add the extension to.

extension

The handle containing the extension data.

idType

A four-byte signature identifying the type of data being added to the `ImageDescription`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows the application to add custom data to an `ImageDescriptionHandle`. This data could be specific to the compressor component referenced by the `ImageDescription` structure.

Special Considerations

The Image Compression Manager makes a copy of the data referred to by the `extension` parameter. Thus, your application should dispose its copy of the data when it is no longer needed.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

AlignScreenRect

Aligns a specified rectangle to the strictest screen that the rectangle intersects.

```
void AlignScreenRect (
   Rect *rp,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

rp

A pointer to a rectangle defined in global screen coordinates.

alignmentProc

Points to your own alignment behavior function. Set this parameter to `NIL` to use the standard behavior.

Discussion

For a specification of your alignment function, see `ICMAlignmentProc`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

AlignWindow

Moves a specified window to the nearest optimal alignment position.

```
void AlignWindow (
   WindowRef wp,
   Boolean front,
   const Rect *alignmentRect,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

wp
: Points to the window to be aligned.

front
: The frontmost window. If the `front` parameter is TRUE and the window specified in the `wp` parameter isn't the active window, `AlignWindow` makes it the active window.

alignmentRect
: A pointer to a rectangle in window coordinates that allows you to align the window to a rectangle within the window. Set this parameter to `NIL` to align using the bounds of the window.

alignmentProc
: Points to a function that allows you to provide your own alignment behavior. Set this parameter to `NIL` to use the standard behavior.

Discussion
For a specification of your alignment function, see `ICMAlignmentProc`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

MakeEffectMovie

MovieGWorlds

QTCarbonShell

SimpleVideoOut

Declared In
`ImageCompression.h`

CDSequenceBusy

Checks the status of an asynchronous compression or decompression operation.

```
OSErr CDSequenceBusy (
   ImageSequence seqID
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 55) or `CompressSequenceBegin` (page 43).

Return Value

If there is no asynchronous operation in progress, `CDSequenceBusy` returns a 0 result code. If there is an asynchronous operation in progress, the result code is 1. Negative result codes indicate an error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceChangedSourceData

Notifies the compressor that the image source data has changed.

```
OSErr CDSequenceChangedSourceData (
   ImageSequenceDataSource sourceID
);
```

Parameters

sourceID

Contains the source identifier of the data source.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to indicate that the image has changed but the data pointer to that image has not changed. For example, if the data pointer points to the base address of a `PixMap` structure, the image in the `PixMap` can change, but the data pointer remains constant.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceDisposeDataSource

Disposes of a data source.

```
OSErr CDSequenceDisposeDataSource (
   ImageSequenceDataSource sourceID
);
```

Parameters

sourceID

The data source identifier that was returned by the `CDSequenceNewDataSource` (page 29) function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to dispose of a data source created by the `CDSequenceNewDataSource` (page 29) function. All data sources are automatically disposed when the sequence they are associated with is disposed.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceDisposeMemory

Disposes of memory allocated by the codec.

```
OSErr CDSequenceDisposeMemory (
   ImageSequence seqID,
   Ptr data
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

data

Points to the previously allocated memory block.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You call this function to release memory allocated by the `CDSequenceNewMemory` (page 31) function.

Special Considerations

Do not call `CDSequenceDisposeMemory` at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceEnd

Indicates the end of processing for an image sequence.

```
OSErr CDSequenceEnd (
   ImageSequence seqID
);
```

Parameters

`seqID`
Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 55) or `CompressSequenceBegin` (page 43).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

qteffects

qteffects.win

SGDataProcSample

VideoProcessing

Declared In
`ImageCompression.h`

CDSequenceEquivalentImageDescription

Reports whether two image descriptions are the same.

```
OSErr CDSequenceEquivalentImageDescription (
   ImageSequence seqID,
   ImageDescriptionHandle newDesc,
   Boolean *equivalent
);
```

Parameters

`seqID`
Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

newDesc

A handle to the `ImageDescription` structure structure that describes the compressed image.

equivalent

A pointer to a Boolean value. If the `ImageDescriptionHandle` provided in the `newDesc` parameter is equivalent to the `ImageDescription` structure currently in use by the image sequence, this value is set to TRUE. If the `ImageDescriptionHandle` is not equivalent, and therefore a new image sequence must be created to display an image using the new image description, this value is set to FALSE.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows an application to ask whether two image descriptions are the same. If they are, the decompressor does not have to create a new image decompression sequence to display those images.

Special Considerations

The Image Compression Manager can only implement part of this function by itself. There are some fields in the `ImageDescription` structure that it knows are irrelevant to the decompressor. If the Image Compression Manager determines that there are differences in fields that may be significant to the codec, it calls the function `ImageCodecIsImageDescriptionEquivalent` to ask the codec.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtactiontargets

qteffects.win

qtsprites.win

qtwiredactions

qtwiredspritesjr.win

Declared In

`ImageCompression.h`

CDSequenceEquivalentImageDescriptionS

Undocumented

```
OSErr CDSequenceEquivalentImageDescriptionS (
   ImageSequence seqID,
   ImageDescriptionHandle newDesc,
   Boolean *equivalent,
   Boolean *canSwitch
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

newDesc

A handle to the `ImageDescription` structure structure that describes the compressed image.

equivalent

A pointer to a Boolean value. If the `ImageDescriptionHandle` provided in the `newDesc` parameter is equivalent to the `ImageDescription` structure currently in use by the image sequence, this value is set to TRUE. If the `ImageDescriptionHandle` is not equivalent, and therefore a new image sequence must be created to display an image using the new image description, this value is set to FALSE.

canSwitch

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceFlush

Stops a decompression sequence, aborting processing of any queued frames.

```
OSErr CDSequenceFlush (
   ImageSequence seqID
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 55).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is used to tell a decompressor component to stop processing of any queued scheduled asynchronous decompression. This is useful when several frames have been queued for decompression in the future and the application needs to suspend playback of the sequence.

For any outstanding frames, your application's completion routine, passed to `DecompressSequenceFrameWhen` (page 60), will be called with an error result of -1, indicating that the frame was cancelled. If any frames are currently being decompressed and cannot be cancelled, `CDSequenceFlush` waits until the frame has finished decompressing before returning.

Version Notes

Introduced in QuickTime 2.0.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceGetDataSource

Gets a data source for a decompression sequence.

```
OSErr CDSequenceGetDataSource (
   ImageSequence seqID,
   ImageSequenceDataSource *sourceID,
   OSType sourceType,
   long sourceInputNumber
);
```

Parameters

`seqID`
The image sequence that this source is associated with.

`sourceID`
A pointer to the source reference identifying this source.

`sourceType`
A four-character code describing how the input will be used. This value is passed by `CDSequenceNewDataSource` (page 29) when the source is created.

`sourceInputNumber`
A value passed by `CDSequenceNewDataSource` (page 29) when the source is created.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceInvalidate

Notifies the Image Compression Manager that the destination port for the given image decompression sequence has been invalidated.

```
OSErr CDSequenceInvalidate (
   ImageSequence seqID,
   RgnHandle invalRgn
);
```

Parameters

`seqID`
Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 55).

invalRgn

A handle to the region specifying the invalid portion of the image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You call this function to force the Image Compression Manager to redraw the screen bits on the next decompression operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceNewDataSource

Creates a new data source.

```
OSErr CDSequenceNewDataSource (
   ImageSequence seqID,
   ImageSequenceDataSource *sourceID,
   OSType sourceType,
   long sourceInputNumber,
   Handle dataDescription,
   ICMConvertDataFormatUPP transferProc,
   void *refCon
);
```

Parameters

seqID

The unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

sourceID

Returns the new data source identifier.

sourceType

A four-character code describing how the input will be used. This code is usually derived from the information returned by the codec. For example, if a mask plane was passed, this field might contain `'mask'`.

sourceInputNumber

More than one instance of a given source type may exist. The first occurrence should have a source input number of 1, the second a source input number of 2, and so on.

dataDescription

A handle to a data structure describing the input data. For compressed image data, this is an `ImageDescriptionHandle`.

`transferProc`

A routine that allows the application to transform the type of the input data to the kind of data preferred by the codec. The client of the codec passes the source data in the form most convenient for it. If the codec needs the data in another form, it can negotiate with the client or directly with the Image Compression Manager to obtain the required data format.

`refCon`

A reference constant to be passed to the transfer procedure. Use this parameter to point to a data structure containing any information your function needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns a `sourceID` parameter which must be passed to all other functions that reference the source. All data sources are automatically disposed when the sequence they are associated with is disposed.

```
// CDSequenceNewDataSource coding example
// See "Discovering QuickTime," page 309
{
    ImageSequenceDataSource     lSrc1 =0;
    // Store a description of the first GWorld in hImageDesc1
    nErr =MakeImageDescriptionForPixMap(GetGWorldPixMap(gWorld1),
                    &hImageDesc1);
    // Create a source from the GWorld description.
    nErr =CDSequenceNewDataSource(gEffectSequenceID,
                                &lSrc1,
                                'srcA',
                                1,
                                (Handle)hImageDesc1,
                                NIL,
                                0);
    // Set the data for source srcA to be the pixMap of gWorld1
    CDSequenceSetSourceData(lSrc1,
                        GetPixBaseAddr(GetGWorldPixMap(gWorld1)),
                        (**hImageDesc1).dataSize);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win

qtshoweffect

qtshoweffect.win

VideoProcessing

vrscript.win

Declared In
`ImageCompression.h`

CDSequenceNewMemory

Requests codec-allocated memory.

```
OSErr CDSequenceNewMemory (
   ImageSequence seqID,
   Ptr *data,
   Size dataSize,
   long dataUse,
   ICMMemoryDisposedUPP memoryGoneProc,
   void *refCon
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

data

Returns a pointer to the allocated memory.

dataSize

The requested size of the data buffer.

dataUse

A code (see below) that indicates how the memory is to be used. For example, the memory may be used to store compressed image or mask plane data, or used as an offscreen image buffer. If there is no benefit to storing a particular kind of data in codec memory, the codec should deny the request for the memory allocation. See these constants:

memoryGoneProc

A pointer to a callback function that will be called before disposing of the memory allocated by a codec, as described in `ICMMemoryDisposedProc`.

refCon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Because many hardware decompression boards contain dedicated on-board memory, significant performance gains can be realized if this memory is used to store data before it is decompressed. When memory is allocated, a callback function must be provided, as described in `ICMMemoryDisposedProc`. The decompressor can dispose of all memory it has allocated at any time, but it calls the callback routine before disposing of the memory. A callback procedure is required because memory on the hardware decompression board may be limited. If the decompressor cannot deallocate memory as required, it is possible that an idle decompressor instance may be holding a large amount of memory, denying those resources to the currently active decompressor instance. When the callback procedure is called, the memory is still available. This allows any pending reads into the block to be canceled before the block is disposed. The decompressor disposing the memory must ensure that it is not disposing a block that it is currently using (that is, a block that contains the currently decompressing frame). To dispose of the memory, use `CDSequenceDisposeMemory` (page 24).

Special Considerations

Decompressor memory must never be disposed at interrupt time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceSetSourceData

Sets data in a new frame to a specific data source.

```
OSErr CDSequenceSetSourceData (
   ImageSequenceDataSource sourceID,
   void *data,
   long dataSize
);
```

Parameters

sourceID

Contains the source identifier of the data source.

data

Points to the data. This pointer must contain a 32-bit clean address.

dataSize

The size of the data buffer.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is called to set data in a new frame to a specific source. For example, as a new frame of compressed data arrives at a source, `CDSequenceSetSourceData` will be called.

```
// CDSequenceSetSourceData coding example
// See "Discovering QuickTime," page 309
{
    ImageSequenceDataSource     lSrc1 =0;
    // Store a description of the first GWorld in hImageDesc1
    nErr =MakeImageDescriptionForPixMap(GetGWorldPixMap(gWorld1),
                    &hImageDesc1);
    // Create a source from the GWorld description.
    nErr =CDSequenceNewDataSource(gEffectSequenceID,
                                    &lSrc1,
                                    'srcA',
                                    1,
                                    (Handle)hImageDesc1,
                                    NIL,
                                    0);
    // Set the data for source srcA to be the pixMap of gWorld1
    CDSequenceSetSourceData(lSrc1,
                            GetPixBaseAddr(GetGWorldPixMap(gWorld1)),
                            (**hImageDesc1).dataSize);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win

qtshoweffect

qtshoweffect.win

VideoProcessing

vrscript.win

Declared In
`ImageCompression.h`

CDSequenceSetSourceDataQueue

Sets a data queue as the source for a decompression sequence.

```
OSErr CDSequenceSetSourceDataQueue (
   ImageSequenceDataSource sourceID,
   QHdrPtr dataQueue
);
```

Parameters

`sourceID`

Contains the source identifier of the data source.

`dataQueue`

A pointer to a `QHdr` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceSetTimeBase

Sets a time base for a decompression sequence.

```
OSErr CDSequenceSetTimeBase (
   ImageSequence seqID,
   void *base
);
```

Parameters

seqID

A unique sequence identifier that was returned by `CompressSequenceBegin` (page 43).

base

A pointer to the time base for this operation. Your application obtains this time base identifier from `NewTimeBase`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When you run a visual effect outside a movie, you must designate a time base that will be used when the effect is run. The following code illustrates this use of `CDSequenceSetTimeBase`:

```
// CDSequenceSetTimeBase coding example
// See "Discovering QuickTime," page 310
timeBase =NewTimeBase();
SetTimeBaseRate(timeBase, 0);
CDSequenceSetTimeBase(gEffectSequenceID, timeBase);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects.win

qtshoweffect

qtshoweffect.win

VideoProcessing

vrscript.win

Declared In

`ImageCompression.h`

CodecManagerVersion

Determines the version of the installed Image Compression Manager.

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

Parameters

version

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.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns the version information as a long integer value. Use this function to retrieve the version number associated with the Image Compression Manager that is installed on a particular computer.

Special Considerations

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.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompAdd

Undocumented

```
void CompAdd (
   wide *src,
   wide *dst
);
```

Parameters

`src`

Undocumented

`dst`

Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompCompare

Undocumented

```
long CompCompare (
   const wide *a,
   const wide *minusb
);
```

Parameters

a

Undocumented

minusb

Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompDiv

Undocumented

```
long CompDiv (
   wide *numerator,
   long denominator,
   long *remainder
);
```

Parameters

numerator

Undocumented

denominator

Undocumented

remainder

Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SoftVDigX

Declared In
`ImageCompression.h`

CompFixMul

Undocumented

```
void CompFixMul (
   wide *compSrc,
   Fixed fixSrc,
   wide *compDst
);
```

Parameters

`compSrc`
: *Undocumented*

`fixSrc`
: *Undocumented*

`compDst`
: *Undocumented*

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompMul

Undocumented

```
void CompMul (
   long src1,
   long src2,
   wide *dst
);
```

Parameters

`src1`
: *Undocumented*

`src2`
: *Undocumented*

`dst`
: *Undocumented*

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompMulDiv

Undocumented

```
void CompMulDiv (
   wide *co,
   long mul,
   long divisor
);
```

Parameters

co
    Undocumented

mul
    Undocumented

divisor
    Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompMulDivTrunc

Undocumented

```
void CompMulDivTrunc (
   wide *co,
   long mul,
   long divisor,
   long *remainder
);
```

Parameters

co
    Undocumented

mul
    Undocumented

divisor
    Undocumented

remainder
    Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompNeg

Undocumented

```
void CompNeg (
   wide *dst
);
```

Parameters

`dst`
: *Undocumented*

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompressImage

Compresses a single-frame image that is currently stored as a pixel map structure.

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

Parameters

`src`
: A handle to the image to be compressed. The image must be stored in a pixel map structure.

`srcRect`
: A pointer to a rectangle defining the portion of the image to compress.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

A compressor type; see `Codec Identifiers`.

desc

A handle that is to receive a formatted `ImageDescription` 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 `GetMaxCompressionSize` (page 104) function. The Image Compression Manager places the actual size of the compressed image into the `dataSize` field of the `ImageDescription` structure structure referred to by the `desc` parameter. This pointer must contain a 32-bit clean address.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The following code sample illustrates the process of compressing and decompressing a pixel map.

```
// CompressImage coding example
// See "Discovering QuickTime," page 286
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                    lMaxCompressedSize =0;
    Handle                  hCompressedData =NIL;
    Ptr                     pCompressedData;
    ImageDescriptionHandle  hImageDesc =NIL;
    OSErr                   nErr;
    PicHandle               hpicPicture =NIL;
    Rect                    rectImage =(**hpmImage).bounds;
    CodecType               dwCodecType =kJPEGCodecType;
    CodecComponent          codec =(CodecComponent)anyCodec;
    CodecQ                  dwSpatialQuality =codecNormalQuality;
    short                   nDepth =0;          // let ICM choose depth
    nErr =GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                                        dwSpatialQuality,
                                        dwCodecType,
                                        (CompressorComponent)codec,
                                        &lMaxCompressedSize);
    if (nErr !=noErr)
        return NIL;

    hImageDesc =(ImageDescriptionHandle)NewHandle(4);
    hCompressedData =NewHandle(lMaxCompressedSize);
    if ((hCompressedData !=NIL) && (hImageDesc !=NIL)) {
        MoveHHi(hCompressedData);
        HLock(hCompressedData);
        pCompressedData =StripAddress(*hCompressedData);

        nErr =CompressImage(hpmImage,
                                &rectImage,
                                dwSpatialQuality,
                                dwCodecType,
                                hImageDesc,
                                pCompressedData);

        if (nErr ==noErr) {
            ClipRect(&rectImage);
            hpicPicture =OpenPicture(&rectImage);
            nErr =DecompressImage(pCompressedData,
                                hImageDesc,
                                hpmImage,
                                &rectImage,
                                &rectImage,
                                srcCopy,
                                NIL);
            ClosePicture();
        }
        if (theErr || (GetHandleSize((Handle)hpicPicture) ==
                                    sizeof(Picture))) {
            KillPicture(hpicPicture);
            hpicPicture =NIL;
        }
    }
    if (hImageDesc !=NIL)
        DisposeHandle((Handle)hImageDesc);
    if (hCompressedData !=NIL)
        DisposeHandle(hCompressedData);
    return hpicPicture;
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CocoaCreateMovie

Fiendishthngs

qteffects.win

qtstreamsplicer.win

ThreadsImportMovie

Declared In
`ImageCompression.h`

CompressPicture

Compresses a single-frame image stored as a picture structure and places the result in another picture.

```
OSErr CompressPicture (
   PicHandle srcPicture,
   PicHandle dstPicture,
   CodecQ quality,
   CodecType cType
);
```

Parameters

srcPicture

A handle to the source image, stored as a picture.

dstPicture

A handle to the destination for the compressed image. The compressor resizes this handle for the result data.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

You must set this parameter to a valid compressor identifier; see `Codec Identifiers`. If the value passed in is 0, or `'raw '`, and the source picture is compressed, the destination picture is created as an uncompressed picture and does not require QuickTime for its display.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function compresses only image data. Any other types of data in the picture, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination picture. This function supports parameters governing image quality and compressor type. The compressor infers the other compression parameters from the image data in the source picture.

Special Considerations

If a picture with multiple pixel maps and other graphical objects is passed, the pixel maps will be compressed individually and the other graphic objects will not be affected. This function does not use the graphics port.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

JPEG File Interchange Format

Declared In

`ImageCompression.h`

CompressPictureFile

Compresses a single-frame image stored as a picture file and places the result in another picture file.

```
OSErr CompressPictureFile (
   short srcRefNum,
   short dstRefNum,
   CodecQ quality,
   CodecType cType
);
```

Parameters

srcRefNum

A file reference number for the source `'PICT'` file.

dstRefNum

A file reference number for the destination `'PICT'` file. Note that the compressor overwrites the contents of the file referred to by `dstRefNum`. You must open this file with write permission. The destination file can be the same as the source file specified by the `srcRefNum` parameter.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

A compressor type. You must set this parameter to a valid compressor type constant; see `Codec Identifiers`. If the value passed in is 0, or `'raw-'`, and the source picture is compressed, the destination picture is created as an uncompressed picture and does not require QuickTime to be displayed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function supports parameters governing image quality and compressor type. The compressor infers the other compression parameters from the image data in the source picture file.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CompressSequenceBegin

Signals the beginning of the process of compressing a sequence of frames.

```
OSErr CompressSequenceBegin (
   ImageSequence *seqID,
   PixMapHandle src,
   PixMapHandle prev,
   const Rect *srcRect,
   const Rect *prevRect,
   short colorDepth,
   CodecType cType,
   CompressorComponent codec,
   CodecQ spatialQuality,
   CodecQ temporalQuality,
   long keyFrameRate,
   CTabHandle ctable,
   CodecFlags flags,
   ImageDescriptionHandle desc
);
```

Parameters

seqID

A pointer to a field to receive the unique identifier for this sequence. You must supply this identifier to all subsequent Image Compression Manager functions that relate to this sequence.

src

A handle to a pixel map that will contain the image to be compressed. The image must be stored in a pixel map structure.

prev

A handle to a pixel map that will contain a previous image. The compressor uses this buffer to store a previous image against which the current image (stored in the pixel map referred to by the `src` parameter) is compared when performing temporal compression. This pixel map must be created at the same depth and with the same color table as the source image. The compressor manages the contents of this pixel map based upon several considerations, such as the key frame rate and the degree of difference between compared images. If you want the compressor to allocate this pixel map or if you do not want to perform temporal compression (that is, you have set the value of the `temporalQuality` parameter to 0), set this parameter to `NIL`.

srcRect

A pointer to a rectangle defining the portion of the image to compress. The compressor applies this rectangle to the image stored in the buffer referred to by the `src` parameter.

prevRect

A pointer to a rectangle defining the portion of the previous image to use for temporal compression. The compressor uses this portion of the previous image as the basis of comparison with the current image. The compressor ignores this parameter if you have not provided a buffer for previous images. This rectangle must be the same size as the source rectangle, which is specified with the `srcRect` parameter.

colorDepth

The depth at which the sequence is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the compressed images. 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` (page 86) function.

cType

You must set this parameter to a valid compressor type constant. See `Codec Identifiers`.

codec

Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance 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. See these constants:

spatialQuality

A pointer to a field containing a constant (see below) that defines the desired compressed image quality. You can change the `value` of this parameter for an active sequence by calling `SetCSequenceQuality` (page 154). See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A pointer to a field containing a constant (see below) that defines the desired temporal quality. This parameter governs the level of compression you desire with respect to information between successive frames in the sequence. Set to 0 if you do not want temporal compression. You can change the `value` of this parameter for an active sequence by calling `SetCSequenceQuality` (page 154).

keyFrameRate

Specifies the maximum number of frames allowed between key frames. The compressor determines the optimum placement for key frames based upon the amount of redundancy between adjacent images in the sequence. Consequently, the compressor may insert key frames more frequently than you have requested. However, the compressor never places fewer key frames than is indicated by the setting of the `keyFrameRate` parameter. The compressor ignores this parameter if you have not requested temporal compression (that is, you have set the `temporalQuality` parameter to 0). If you pass in 0 in this parameter, this indicates that there are no key frames in the sequence. If you pass in any other number, it specifies the number of non-key frames between key frames. Set this parameter to 1 to specify all key frames, to 2 to specify every other frame as a key frame, to 3 to specify every third frame as a key frame, and so forth. Your application may change the key frame rate for an active sequence by calling `SetCSequenceKeyFrameRate` (page 152).

ctable

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 `colorDepth` 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 `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` 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 `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

`flags`

Contains flags (see below) that provide further control information. You must set either `codecFlagUpdatePrevious` or `codecFlagUpdatePreviousComp` to 1. Set unused flags to 0. See these constants:

`codecFlagUpdatePrevious`

`codecFlagUpdatePreviousComp`

`codecFlagWasCompressed`

`desc`

A handle that is to receive a formatted `ImageDescription` structure. The Image Compression Manager resizes this handle for the returned image description structure. Your application should store this image description with the compressed sequence. During the compression operation, the Image Compression Manager and the compressor component update the contents of this image description. Consequently, you should not store the image description until the sequence has been completely processed.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The Image Compression Manager prepares for a sequence-compression operation by reserving appropriate system resources. Hence you must call `CompressSequenceBegin` before you call `CompressSequenceFrame` (page 46).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

DesktopSprites

qteffects.win

qtwiredactions

qtwiredspritesjr.win

Declared In
`ImageCompression.h`

CompressSequenceFrame

Compresses one of a sequence of frames.

```
OSErr CompressSequenceFrame (
   ImageSequence seqID,
   PixMapHandle src,
   const Rect *srcRect,
   CodecFlags flags,
   Ptr data,
   long *dataSize,
   UInt8 *similarity,
   ICMCompletionProcRecordPtr asyncCompletionProc
);
```

Parameters

seqID

Unique sequence identifier that was returned by `CompressSequenceBegin` (page 43).

src

A handle to a pixel map that contains the image to be compressed. The image must be stored in a pixel map structure.

srcRect

A pointer to a rectangle defining the portion of the image to compress. The compressor applies this rectangle to the image stored in the buffer referred to by the `src` parameter.

flags

Specifies flags (see below) that provide further control information. You must set either `codecFlagUpdatePrevious` or `codecFlagUpdatePreviousComp` to 1. Set unused flags to 0. See these constants:

`codecFlagUpdatePrevious`

`codecFlagWasCompressed`

`codecFlagUpdatePreviousComp`

`codecFlagForceKeyFrame`

`codecFlagLiveGrab`

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 `GetMaxCompressionSize` (page 104) function. The Image Compression Manager places the actual size of the compressed image into the field referred to by the `dataSize` parameter. This pointer must contain a 32-bit clean address.

dataSize

A pointer to a field that is to receive the size, in bytes, of the compressed image.

similarity

A pointer to a field that is to receive a similarity value. The `CompressSequenceFrame` function returns a value that indicates the similarity of the current frame to the previous frame. A value of 0 indicates that the current frame is a key frame in the sequence. A value of 255 indicates that the current frame is identical to the previous frame. Values from 1 through 254 indicate relative similarity, ranging from very different (1) to very similar (254).

asyncCompletionProc

Points to an `ICMCompletionProc` callback. The compressor calls your completion function when an asynchronous compression operation is complete. You can cause the compression to be performed asynchronously by specifying a completion function if the compressor supports asynchronous compression.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The Image Compression Manager prepares for a sequence-compression operation by reserving appropriate system resources. Hence you must call `CompressSequenceBegin` (page 43) before you call `CompressSequenceFrame`.

Special Considerations

If you specify asynchronous operation, you must not read the compressed data until the compressor indicates that the operation is complete by calling your completion function. Set `asyncCompletionProc` to `NIL` to specify synchronous compression. If you set `asyncCompletionProc` to -1, the operation is performed asynchronously but the compressor does not call your completion function. If the `asyncCompletionProc` parameter is not `NIL`, the following conditions are in effect: the pixels in the source image must stay valid until the completion function is called with its `codecCompletionSource` flag, and the resulting compressed data is not valid until it is called with its `codecCompletionDest` flag set.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

DesktopSprites

qteffects.win

qtwiredactions

qtwiredspritesjr.win

Declared In
`ImageCompression.h`

CompShift

Undocumented

```
void CompShift (
   wide *src,
   short shift
);
```

Parameters

`src`

  Undocumented

`shift`

  Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompSquareRoot

Undocumented

```
unsigned long CompSquareRoot (
   const wide *src
);
```

Parameters

`src`
: *Undocumented*

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompSub

Undocumented

```
void CompSub (
   wide *src,
   wide *dst
);
```

Parameters

`src`
: *Undocumented*

`dst`
: *Undocumented*

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Dimmer2Effect

Dimmer2Effect.win

Declared In
`ImageCompression.h`

ConcatMatrix

Concatenates two matrices, combining the transformations described by both matrices into a single matrix.

```
void ConcatMatrix (
   const MatrixRecord *a,
   MatrixRecord *b
);
```

Parameters

a

A pointer to the source matrix.

b

A pointer to the destination matrix. The `ConcatMatrix` function performs a matrix multiplication operation on the two matrices and leaves the result in the matrix specified by this parameter.

Discussion

This is a matrix multiplication operation, as a result of which [B] =[B] x [A]. Note that matrix multiplication is not commutative.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

Declared In

`ImageCompression.h`

ConvertImage

Converts the format of a compressed image.

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

Parameters

srcDD

A handle to the `ImageDescription` structure that describes the compressed image.

srcData

Points to the compressed image data. This pointer must contain a 32-bit clean address.

colorDepth

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 `GetCodecInfo` (page 86) function.

ctable

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 `colorDepth` 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 `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` 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 `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source `ImageDescription` structure.

accuracy

A constant (see below) that defines the accuracy desired in the decompressed image. Values for this parameter are on the same scale as compression quality. For a good display of still images, you should specify at least `codecHighQuality`. See these constants:

`codecMinQuality`
`codecLowQuality`
`codecNormalQuality`
`codecHighQuality`
`codecMaxQuality`
`codecLosslessQuality`

quality

A constant (see below) that defines the desired compressed image quality.

cType

A compressor type; see `Codec Identifiers`.

codec

Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special constant (see below). Specifying a component instance 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. See these constants:

dstDD

A handle that is to receive a formatted `ImageDescription` structure. The Image Compression Manager resizes this handle for the returned `ImageDescription` 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 `GetMaxCompressionSize` (page 104). 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.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The action of this function is essentially equivalent to decompressing and recompressing the image. During the decompression operation, the decompressor uses the `srcDD`, `srcData`, and accuracy parameters. During the subsequent compression operation, the compressor uses the `colorDepth`, `ctable`, `cType`, codec, quality, `dstDD`, and `dstData` parameters.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CopyMatrix

Copies the contents of one matrix into another matrix.

```
void CopyMatrix (
   const MatrixRecord *m1,
   MatrixRecord *m2
);
```

Parameters

`m1`

The source matrix for the copy operation.

`m2`

A pointer to the destination matrix for the copy operation. `CopyMatrix` copies the values from the matrix specified by the m1 parameter into this matrix.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CountImageDescriptionExtensionType

Counts the number of extensions of a given type in an ImageDescriptionHandle.

```
OSErr CountImageDescriptionExtensionType (
   ImageDescriptionHandle desc,
   long idType,
   long *count
);
```

Parameters

desc

A handle to the `ImageDescription` structure with the extensions to be counted.

idType

Indicates the type of extension to be counted in the specified `ImageDescription` structure. Set the `value` of this parameter to 0 to match any extension, and return a count of all of the extensions.

count

A pointer to an integer that indicates how many extensions of the given type are in the given `ImageDescription` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When used with `GetNextImageDescriptionExtensionType` (page 106), this function allows the application to determine the total set of extensions present in the `ImageDescription` structure designated by `desc`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

DecompressImage

Decompresses a single-frame image into a pixel map structure.

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

Parameters

data

Points to the compressed image data. This pointer must contain a 32-bit clean address.

desc

A handle to the `ImageDescription` structure that describes the compressed image.

`dst`

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`

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 `((**desc).width,(**desc).height)`. If you want to decompress the entire source image, set this parameter to `NIL`. If the parameter is `NIL`, the rectangle is set to the rectangle structure of the `ImageDescription` structure.

`dstRect`

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.

`mode`

The transfer mode for the operation, as listed in `Graphics Transfer Modes`.

`mask`

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 `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Note that `DecompressImage` is invoked through the `StdPix` (page 166) function. The following code sample illustrates the process of compressing and decompressing a pixel map.

```
// DecompressImage coding example
// See "Discovering QuickTime," page 286
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                    lMaxCompressedSize =0;
    Handle                  hCompressedData =NIL;
    Ptr                     pCompressedData;
    ImageDescriptionHandle  hImageDesc =NIL;
    OSErr                   nErr;
    PicHandle               hpicPicture =NIL;
    Rect                    rectImage =(**hpmImage).bounds;
    CodecType               dwCodecType =kJPEGCodecType;
    CodecComponent          codec =(CodecComponent)anyCodec;
    CodecQ                  dwSpatialQuality =codecNormalQuality;
    short                   nDepth =0;          // let ICM choose depth
    nErr =GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                                dwSpatialQuality,
                                dwCodecType,
                                (CompressorComponent)codec,
                                &lMaxCompressedSize);
    if (nErr !=noErr)
        return NIL;

    hImageDesc =(ImageDescriptionHandle)NewHandle(4);
    hCompressedData =NewHandle(lMaxCompressedSize);
    if ((hCompressedData !=NIL) && (hImageDesc !=NIL)) {
        MoveHHi(hCompressedData);
        HLock(hCompressedData);
        pCompressedData =StripAddress(*hCompressedData);
        nErr =CompressImage(hpmImage,
                                        &rectImage,
                                        dwSpatialQuality,
                                        dwCodecType,
                                        hImageDesc,
                                        pCompressedData);

        if (nErr ==noErr) {
            ClipRect(&rectImage);
            hpicPicture =OpenPicture(&rectImage);
            nErr =DecompressImage(pCompressedData,
                                        hImageDesc,
                                        hpmImage,
                                        &rectImage,
                                        &rectImage,
                                        srcCopy,
                                        NIL);
            ClosePicture();
        }
        if (theErr || (GetHandleSize((Handle)hpicPicture) ==
                                            sizeof(Picture))) {
            KillPicture(hpicPicture);
            hpicPicture =NIL;
        }
    }
    if (hImageDesc !=NIL)
        DisposeHandle((Handle)hImageDesc);
    if (hCompressedData !=NIL)
        DisposeHandle(hCompressedData);
    return hpicPicture;
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

FastDitherUsingQT

qteffects.win

qtwiredactions

qtwiredspritesjr.win

Declared In
`ImageCompression.h`

DecompressSequenceBegin

Obsolete. See DecompressSequenceBeginS.

```
OSErr DecompressSequenceBegin (
   ImageSequence *seqID,
   ImageDescriptionHandle desc,
   CGrafPtr port,
   GDHandle gdh,
   const Rect *srcRect,
   MatrixRecordPtr matrix,
   short mode,
   RgnHandle mask,
   CodecFlags flags,
   CodecQ accuracy,
   DecompressorComponent codec
);
```

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Cocoa - SGDataProc

ImproveYourImage

SGDataProcSample

VideoProcessing

vrmovies.win

Declared In
`ImageCompression.h`

DecompressSequenceBeginS

Sends a sample image to a decompressor.

```
OSErr DecompressSequenceBeginS (
   ImageSequence *seqID,
   ImageDescriptionHandle desc,
   Ptr data,
   long dataSize,
   CGrafPtr port,
   GDHandle gdh,
   const Rect *srcRect,
   MatrixRecordPtr matrix,
   short mode,
   RgnHandle mask,
   CodecFlags flags,
   CodecQ accuracy,
   DecompressorComponent codec
);
```

Parameters

seqID

A pointer to a field to receive the unique identifier for the sequence you are creating. You should use this identifier for subsequent calls relating to this decompression sequence.

desc

A handle to the `ImageDescription` structure that describes the compressed image.

data

Points to the compressed image data. This pointer must contain a 32-bit clean address. Ideally, you should pass a pointer to the first frame of the compressed image data, which lets the Image Compression Manager do a better job of preflighting the decompression sequence. If the image data is not available at the time of this call, you can pass `NIL` for this parameter and 0 for `dataSize`. If you pass `NIL` here, then your first call to `DecompressSequenceFrameWhen` (page 60) may require more setup time.

dataSize

The size of the data buffer, or 0 if you passed `NIL` in the `data` parameter.

port

Points to the `CGrafPort` structure for the destination image.

gdh

A handle to the `GDevice` structure for the destination image. You can pass `NIL` if the `GDevice` is implicit in the port selection (for example, if it is an offscreen graphics world).

srcRect

A pointer to a `Rect` structure that defines the portions of the image to decompress. Pass `NIL` if you want to decompress the entire source image. You can call `SetDSequenceSrcRect` (page 161) to change the source rectangle for an active decompression sequence.

matrix

Points to a `MatrixRecord` structure that specifies how to transform the image during decompression. Pass `NIL` to use the identity matrix. Your application can change the matrix for an active sequence by calling `SetDSequenceMatrix` (page 158).

mode

The transfer mode for the operation. See `Graphics Transfer Modes`. Your application can change the transfer mode for an active sequence by calling `SetDSequenceTransferMode` (page 162).

mask

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 `NIL`. Your application can change the clipping mask for an active sequence by calling `SetDSequenceMask` (page 158).

flags

Buffer allocation flags (see below). See these constants:

`codecFlagUseScreenBuffer`

`codecFlagUseImageBuffer`

accuracy

A constant (see below) that defines the desired compression accuracy. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

codec

A decompressor identifier. Specify a particular decompressor by setting this parameter to its identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance 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. See these constants:

Return Value

See `Error Codes`. Returns `codecWouldOffscreenErr` if `codecFlagDontUseImageBuffer` is set and the codec requires an offscreen buffer to decompress to the destination port. Returns `noErr` if there is no error.

Discussion

This function lets you pass a compressed sample so a codec can perform preflighting before the first `DecompressSequenceFrameWhen` (page 60) call. To decompress a series of images, call it once to preflight the decompressor, make calls to `DecompressSequenceFrameWhen` to decompress each image in the sequence, then call `CDSequenceEnd` (page 25) when you are done.

Version Notes

Introduced in QuickTime 1.6.1.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BrideOfMungGrab

MungSaver

qteffects.win

Quartz Composer QCTV

VideoProcessing

Declared In

`ImageCompression.h`

DecompressSequenceFrame

Obsolete. See DecompressSequenceFrameS.

```
OSErr DecompressSequenceFrame (
   ImageSequence seqID,
   Ptr data,
   CodecFlags inFlags,
   CodecFlags *outFlags,
   ICMCompletionProcRecordPtr asyncCompletionProc
);
```

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

ConvertToMovieJr

Inside Mac ICM Code

SGDataProcSample

VideoProcessing

Declared In
`ImageCompression.h`

DecompressSequenceFrameS

Queues a frame for decompression and specifies the size of the compressed data; new applications should use DecompressSequenceFrameWhen.

```
OSErr DecompressSequenceFrameS (
   ImageSequence seqID,
   Ptr data,
   long dataSize,
   CodecFlags inFlags,
   CodecFlags *outFlags,
   ICMCompletionProcRecordPtr asyncCompletionProc
);
```

Parameters

seqID
Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

data
Points to the compressed image data. This pointer must contain a 32-bit clean address.

dataSize
The size of the data buffer.

inFlags
Contains flags (see below) that provide further control information. See these constants:
`codecFlagNoScreenUpdate`
`codecFlagDontOffscreen`
`codecFlagOnlyScreenUpdate`

outFlags
Contains status flags (see below). The decompressor updates these flags at the end of the decompression operation. See these constants:
`codecFlagUsedNewImageBuffer`
`codecFlagUsedImageBuffer`
`codecFlagDontUseNewImageBuffer`
`codecFlagInterlaceUpdate`
`codecFlagCatchUpDiff`

asyncCompletionProc
Points to an `ICMCompletionProcRecord` structure. The compressor calls your completion function when an asynchronous decompression operation is complete. You can cause the decompression to be performed asynchronously by specifying a completion function. If you specify asynchronous operation, you must not read the decompressed image until the decompressor indicates that the operation is complete by calling your completion function. Set `asyncCompletionProc` to `NIL` to specify synchronous decompression. If you set `asyncCompletionProc` to -1, the operation is performed asynchronously but the decompressor does not call your completion function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function accepts the same parameters as the `DecompressSequenceFrame` (page 58) function, with the addition of the `dataSize` parameter.

Special Considerations

New applications should use `DecompressSequenceFrameWhen`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Cocoa - SGDataProc

ImproveYourImage

MungSaver

SGDataProcSample

VideoProcessing

Declared In

`ImageCompression.h`

DecompressSequenceFrameWhen

Queues a frame for decompression and specifies the time at which decompression will begin.

```
OSErr DecompressSequenceFrameWhen (
   ImageSequence seqID,
   Ptr data,
   long dataSize,
   CodecFlags inFlags,
   CodecFlags *outFlags,
   ICMCompletionProcRecordPtr asyncCompletionProc,
   const ICMFrameTimeRecord *frameTime
);
```

Parameters

`seqID`

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

`data`

Points to the compressed image data. This pointer must contain a 32-bit clean address.

`dataSize`

The size of the data buffer.

`inFlags`

Contains flags (see below) that provide further control information. See these constants:

`codecFlagNoScreenUpdate`

`codecFlagDontOffscreen`

`codecFlagOnlyScreenUpdate`

outFlags

Contains status flags (see below). The decompressor updates these flags at the end of the decompression operation. See these constants:

`codecFlagUsedNewImageBuffer`

`codecFlagUsedImageBuffer`

`codecFlagDontUseNewImageBuffer`

`codecFlagInterlaceUpdate`

`codecFlagCatchUpDiff`

asyncCompletionProc

Points to an `ICMCompletionProcRecord` structure. The compressor calls your completion function when an asynchronous decompression operation is complete. You can cause the decompression to be performed asynchronously by specifying a completion function. If you specify asynchronous operation, you must not read the decompressed image until the decompressor indicates that the operation is complete by calling your completion function. Set `asyncCompletionProc` to `NIL` to specify synchronous decompression. If you set `asyncCompletionProc` to -1, the operation is performed asynchronously but the decompressor does not call your completion function.

frameTime

Points to an `ICMFrameTimeRecord` structure, which contains the frame's time information, including the time at which the frame should be displayed, its duration, and the movie's playback rate. This parameter can be `NIL`, in which case the decompression operation will happen immediately.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The following code snippet shows this function being used to execute one frame of a visual effect.

```
// DecompressSequenceFrameWhen coding example
// See "Discovering QuickTime," page 310
// Decompress a single step of the effect sequence.
OSErr RunEffect(TimeValue lTime, int nNumberOfSteps)
{
    OSErr               nErr =noErr;
    ICMFrameTimeRecord  ftr;
    // Set the timebase time to the step of the sequence to be rendered
    SetTimeBaseValue(timeBase, lTime, nNumberOfSteps);
    ftr.value.lo          =lTime;
    ftr.value.hi          =0;
    ftr.scale             =nNumberOfSteps;
    ftr.base              =0;
    ftr.duration          =nNumberOfSteps;
    ftr.rate              =0;
    ftr.recordSize        =sizeof(ftr);
    ftr.frameNumber       =1;
    ftr.flags             =icmFrameTimeHasVirtualStartTimeAndDuration;
    ftr.virtualStartTime.lo     =0;
    ftr.virtualStartTime.hi     =0;
    ftr.virtualDuration         =nNumberOfSteps;
    HLock(hEffectDesc);
    DecompressSequenceFrameWhen(gEffectSequenceID,
                                StripAddress(*hEffectDesc),
                                GetHandleSize(hEffectDesc),
                                0,
                                0,
                                NIL,
                                &ftr);
    HUnlock(hEffectDesc);
}
```

Special Considerations

If the current decompressor component does not support scheduled asynchronous decompression, the Image Compression Manager returns an error code of `codecCantWhenErr`. In this case, the application will need to reissue the request with the `frameTime` parameter set to `NIL`. If the decompressor cannot service your request at a particular time (for example, if its queue is full), the Image Compression Manager returns an error code of `codecCantQueueErr`. The best way to determine whether a decompressor component supports this function is to call the function and test the result code. A decompressor's ability to honor the request may change based on screen depth, clipping settings, and so on.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win

qtshoweffect

qtshoweffect.win

VideoProcessing

vrscript.win

Declared In
`ImageCompression.h`

DisposeCodecNameList

Disposes of the compressor name list structure you obtained by calling GetCodecNameList.

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

Parameters

`list`

Points to the compressor name list to be disposed of. You obtain the compressor list by calling `GetCodecNameList` (page 86).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DragAlignedGrayRgn

Drags a specified gray region along an optimal alignment grid.

```
long DragAlignedGrayRgn (
   RgnHandle theRgn,
   Point startPt,
   Rect *boundsRect,
   Rect *slopRect,
   short axis,
   UniversalProcPtr actionProc,
   Rect *alignmentRect,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

theRgn

A handle to the specified region for this operation. When the user holds down the mouse button, `DragAlignedGrayRgn` pulls a gray outline of the region around following the movement of the mouse until the mouse button is released.

startPt

The point where the mouse button was originally pressed in the local coordinates of the current graphics port.

boundsRect

A pointer to the boundary rectangle of the current graphics port. The offset point follows the mouse location except that `DragAlignedGrayRgn` never moves the offset point outside this rectangle. This limits the travel of the region's outline, not the movements of the mouse.

slopRect

A pointer to the slop rectangle that completely encloses the boundary rectangle so that the user is allowed some flexibility in moving the mouse.

axis

Allows you to constrain the region's motion to only one axis (see constants below). See these constants:

actionProc

Points to a function that defines some action to be performed repeatedly as long as the user holds down the mouse button. The function should have no parameters. If the `actionProc` parameter is `NIL`, `DragAlignedGrayRgn` simply retains control until the mouse button is released.

alignmentRect

A pointer to a rectangle within the bounds of the region specified in the parameter `theRgn`. Pass `NIL` to align using the `bounds` of the parameter `theRgn`.

alignmentProc

A pointer to your alignment behavior function; see `ICMAlignmentProc`. Pass `NIL` to use the standard behavior.

Return Value

The difference between the point where the mouse button was pressed and the offset point; that is, the point in the region whose horizontal and vertical offsets from the upper-left corner of the region's enclosing rectangle are the same as the offsets of the starting point when the user pressed the mouse button. The vertical difference between the starting point and the offset point is stored in the high-order word of the return value and the horizontal difference is stored in the low-order word.

Discussion

This function limits the movement of the region defined by `theRgn` according to the constraints set by the `boundsRect` and `slopRect` parameters. While the cursor is inside the `boundsRect` rectangle, the region's outline follows it normally. If the mouse button is released while the cursor is within this rectangle, the return value reflects the simple distance that the cursor moved in each dimension. When the cursor moves outside the `boundsRect` rectangle, the offset point stops at the edge of the `boundsRect` rectangle. If the mouse button is released while the cursor is outside the `boundsRect` rectangle but inside the `slopRect` rectangle, the return value reflects only the difference between the starting point and the offset point, regardless of how far outside of the `boundsRect` rectangle the cursor may have moved. (Note that part of the region can fall outside the `boundsRect` rectangle, but not the offset point.) When the cursor moves outside the `slopRect` rectangle, the region's outline disappears from the screen. `DragAlignedGrayRgn` continues to track the cursor, however, and if the cursor moves back into the `slopRect` rectangle, the outline reappears. If the mouse button is released while the cursor is anywhere inside the `slopRect` rectangle, the window is redrawn in its new location, calculated from the value returned by `DragAlignedGrayRgn` If the mouse button is released while the cursor is outside the `slopRect` rectangle, both words of the return value are set to 0x8000 and the window does not move from its original location.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

DragAlignedWindow

Drags the specified window along an optimal alignment grid.

```
void DragAlignedWindow (
   WindowRef wp,
   Point startPt,
   Rect *boundsRect,
   Rect *alignmentRect,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

wp

A window pointer to the window to be dragged.

startPt

A point that is equal to the point where the mouse button was pressed (in global coordinates, as stored in the `where` field of the event structure). `DragAlignedWindow` pulls a gray outline of the window around the screen, following the movements of the mouse until the button is released.

boundsRect

Points to the boundary rectangle in global coordinates. If the mouse button is released when the mouse position is outside the limits of the boundary rectangle, `DragAlignedWindow` returns without moving the window or making it the active window. For a document window, the boundary rectangle typically is four pixels in from the menu bar and from the other edges of the screen, to ensure that there won't be less than a four-pixel-square area of the title bar visible on the screen.

alignmentRect

Points to a rectangle in window coordinates that allows you to align the window to a rectangle within the window. Set this parameter to `NIL` to align using the bounds of the window.

alignmentProc

A pointer to your alignment behavior function; see `ICMAlignmentProc`. Pass `NIL` to use the standard behavior.

Discussion

The following code sample illustrates the use of `DragAlignedWindow`:

```
// DragAlignedWindow coding example
// See "Discovering QuickTime," page 265
Boolean IsQuickTimeInstalled (void)
{
    OSErr       nErr;
    long        lResult;
    nErr =Gestalt(gestaltQuickTime, &lResult);
    return (nErr ==noErr);
}
void MyInitialize (void)
{
    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NIL);
    MaxApplZone();
    EnterMovies();
}
WindowRef MakeMyWindow (void)
{
    WindowRef   pMacWnd;
    Rect        rectWnd ={0, 0, 120, 160};
    Rect        rectBest;
    // figure out the best monitor for the window
    GetBestDeviceRect(NIL, &rectBest);
    // put the window in the top left corner of that monitor
    OffsetRect(&rectWnd, rectBest.left + 10, rectBest.top + 50);
    // create the window
    pMacWnd =NewCWindow(NIL, &rectWnd, "\pGrabber",
                        TRUE, noGrowDocProc, (WindowRef)-1,
                        TRUE, 0);
    // set the port to the new window
    SetPort(pMacWnd);
    return pMacWnd;
}
main (void)
{
    WindowRef           pMacWnd;
    SeqGrabComponent    seqGrab;
    SGChannel           sgchanVideo, sgchanSound;
    Boolean             bDone =FALSE;
    OSErr               nErr;
    MyInitialize();
    pMacWnd =MakeMyWindow();
    seqGrab =MakeMySequenceGrabber(pMacWnd);
    if (seqGrab ==NIL)
```

```
        return;
    MakeMyGrabChannels(seqGrab, &sgchanVideo, &sgchanSound,
                    &pMacWnd->
portRect, FALSE);
    nErr =SGStartPreview(seqGrab);
    while (!bDone) {
        ICMAlignmentProcRecord  apr;
        short                   nPart;
        WindowRef               pWhichWnd;
        EventRecord             er;
        GetNextEvent(everyEvent, &er);
        switch (er.what) {
            case nullEvent:     // give the sequence grabber time
                nErr =SGIdle(seqGrab);
                if (nErr !=noErr)
                    bDone =TRUE;
                break;
            case updateEvt:
                if (er.message ==(long)pMacWnd) {
                    // inform the sequence grabber of the update
                    SGUpdate(seqGrab,((WindowPeek)
                                 pMacWnd)->
updateRgn);
                    // and swallow the update event
                    BeginUpdate(pMacWnd);
                    EndUpdate(pMacWnd);
                }
                break;
            case mouseDown:
                nPart =FindWindow(er.where, &pWhichWnd);
                if (pWhichWnd !=pMacWnd)
                    break;
                switch (nPart) {
                    case inContent:
                        // pause until mouse button is released
                        SGPause(seqGrab, TRUE);
                        while (StillDown())
                            SGPause(seqGrab, FALSE);
                            break;
                    case inGoAway:
                        bDone =TrackGoAway(pMacWnd, er.where);
                        break;
                    case inDrag:
                        // pause when dragging window so video
                        // doesn't draw in the wrong place
                        SGPause(seqGrab, TRUE);
                        SGGetAlignmentProc(seqGrab, &apr);
                        DragAlignedWindow(pMacWnd,
                                    er.where,
                                    &screenBits.bounds,
                                    NIL, &alignProc);
                        SGPause(seqGrab, FALSE);
                        break;
                }
                break;
        }
    }
    // clean up
    SGStop(seqGrab);
    CloseComponent(seqGrab);
    DisposeWindow(pMacWnd);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

Movie From DataRef

MovieBrowser

SGDataProcSample

vrbackbuffer

Declared In
`ImageCompression.h`

DrawPictureFile

Draws an image from a specified picture file in the current graphics port.

```
OSErr DrawPictureFile (
   short refNum,
   const Rect *frame,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

refNum

A file reference number for the source PICT file.

frame

A pointer to the rectangle into which the image is to be loaded. The compressor scales the source image to fit into this destination rectangle.

progressProc

Points to an `ICMProgressProc` callback. During the operation, the draw function may occasionally call a function you provide in order to report its progress; see `ICMProgressProcRecord`. If you have not provided a progress function, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function draws the picture that it finds in the picture file specified by the `refNum` parameter within the rectangle specified by the `frame` parameter. The Image Compression Manager performs any spooling that may be necessary when reading the picture file. Specify a clipping region appropriate for your picture before drawing it. If the clipping region is very large (as it is when a graphics port is initialized) and you want to scale the picture, the clipping region can become invalid when `DrawPictureFile` scales the clipping region,

in which case your picture will not be drawn. On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when `DrawPictureFile` draws it. Setting a clipping region equal to the port rectangle of the current graphics port always sets a valid clipping region.

Special Considerations

When it scales fonts, `DrawPictureFile` changes the size of the font instead of scaling the bits. However, the widths used by `bitmap` fonts are not always linear. For example, the 12-point width isn't exactly 1/2 of the 24-point width. This can cause lines of text to become slightly longer or shorter as the picture is scaled. The easiest way to avoid such problems is to specify a destination rectangle that is the same size as the bounding rectangle for the picture.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
JPEG File Interchange Format

Declared In
`ImageCompression.h`

DrawTrimmedPicture

Draws an image that is stored as a picture into the current graphics port and trims that image to fit a specified region.

```
OSErr DrawTrimmedPicture (
   PicHandle srcPicture,
   const Rect *frame,
   RgnHandle trimMask,
   short doDither,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

`srcPicture`
: A handle to the source image, stored as a picture.

`frame`
: A pointer to the rectangle into which the decompressed image is to be loaded.

`trimMask`
: A handle to a clipping region in the destination coordinate system. The decompressor applies this mask to the destination image and ignores any image data that fall outside the specified region. Set this parameter to `NIL` if you do not want to clip the source image.

`doDither`
: Indicates whether to dither the image. Use this parameter if you want the image to be dithered when it is displayed on a lower-resolution screen (see below). See these constants:

    `defaultDither`
    `forceDither`
    `suppressDither`

progressProc

A pointer to an `ICMProgressProc` callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress function, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function works with compressed image data; the source data stays compressed. The function trims the image to fit the specified clipping region. Note that if you just use a clip while making a picture, the data (though not visible) is still stored in the picture.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DrawTextCodec

Declared In

`ImageCompression.h`

DrawTrimmedPictureFile

Draws an image that is stored as a picture file into the current graphics port and trims that image to fit a specified region.

```
OSErr DrawTrimmedPictureFile (
   short srcRefnum,
   const Rect *frame,
   RgnHandle trimMask,
   short doDither,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

srcRefnum

A file reference number for the source PICT file.

frame

A pointer to the rectangle into which the decompressed image is to be loaded.

trimMask

A handle to a clipping region in the destination coordinate system. The decompressor applies this mask to the destination image and ignores any image data that fall outside the specified region. Set this parameter to `NIL` if you do not want to clip the source image. In this case, this function acts like `DrawPictureFile` (page 67).

doDither

Indicates whether to dither the image. Use this parameter if you want the image to be dithered when it is displayed on a lower-resolution screen (see below). See these constants:

`defaultDither`

`forceDither`

`suppressDither`

progressProc

A pointer to an `ICMProgressProc` callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress function, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to save part of a picture, since the image data that is not within the trim region is ignored and is not included in the destination picture file. All the remaining objects in the resulting object are clipped.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DrawTextCodec

Declared In

`ImageCompression.h`

EqualMatrix

Compares two matrices and returns a result that indicates whether the matrices are equal.

```
Boolean EqualMatrix (
   const MatrixRecord *m1,
   const MatrixRecord *m2
);
```

Parameters

m1

A pointer to one matrix for the compare operation.

m2

A pointer to the other matrix for the compare operation.

Return Value

TRUE if the matrices are equal, FALSE otherwise.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qdmediahandler

qdmediahandler.win

Declared In
`ImageCompression.h`

FCompressImage

Compresses a single-frame image that is currently stored as a pixel map structure, with added control over the compression process.

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

Parameters

`src`

A handle to the image to be compressed. The image must be stored in a pixel map structure.

`srcRect`

A pointer to a rectangle defining the portion of the image to compress.

`colorDepth`

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` (page 86) function.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

A compressor type. You must set this parameter to a valid compressor type constant.

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance 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. See these constants:

ctable

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 `colorDepth` 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 `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` 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 `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

flags

Contains a flag (see below) that indicates whether or not the image was previously compressed. See these constants:

`codecFlagWasCompressed`

bufferSize

The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If you have not specified a data-unloading function, set this parameter to 0.

flushProc

Points to an `ICMDataProc` data-unloading callback. 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. If you have not provided a data-unloading callback, set this parameter to `NIL`. In this case, the compressor writes the entire compressed image into the memory location specified by the `data` parameter.

progressProc

Points to an `ICMProgressProc` progress callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

desc

A handle that is to receive a formatted `ImageDescription` structure. The Image Compression Manager resizes this handle for the returned `ImageDescription` 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 `GetMaxCompressionSize` (page 104) function. 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. The Image Compression Manager places the actual size of the compressed image into the `dataSize` field of the `ImageDescription` structure referenced by the `desc` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function acts like `CompressImage` (page 39), but gives your application additional control over the parameters that guide the compression operation.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

VelEng Wavelet

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In
`ImageCompression.h`

FCompressPicture

Compresses a single-frame image stored as a picture structure and places the result in another picture, with added control over the compression process.

```
OSErr FCompressPicture (
   PicHandle srcPicture,
   PicHandle dstPicture,
   short colorDepth,
   CTabHandle ctable,
   CodecQ quality,
   short doDither,
   short compressAgain,
   ICMProgressProcRecordPtr progressProc,
   CodecType cType,
   CompressorComponent codec
);
```

Parameters

srcPicture

A handle to the source image, stored as a picture.

dstPicture
A handle to the destination for the compressed image. The compressor resizes this handle for the result data.

colorDepth
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 `GetCodecInfo` (page 86) function.

ctable
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 `colorDepth` 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 `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` 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 `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

quality
A constant that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

doDither
A constant (see below) that indicates whether to dither the image. Use this parameter to indicate whether you want the image to be dithered when it is displayed on a lower-resolution screen. See these constants:

`defaultDither`

`forceDither`

`suppressDither`

compressAgain
Indicates whether to recompress compressed image data in the `picture`. Use this parameter to control whether any compressed image data that is in the source picture should be decompressed and then recompressed using the current parameters. Set the `value` of this parameter to TRUE to recompress such data. Set the `value` of the parameter to FALSE to leave the data as it is. Note that recompressing the data may have undesirable side effects, including image quality degradation.

progressProc
Points to an `ICMProgressProc` callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

cType

A compressor type. You must set this parameter to a valid compressor type constant; see `Codec Identifiers`. If the value passed in is 0, or `'raw '`, the resulting picture is not compressed and does not require QuickTime to be displayed.

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If a picture with multiple pixel maps and other graphical objects is passed, the pixel maps will be compressed individually and the other graphic objects will not be affected. `FCompressPicture` compresses only image data. Any other types of data in the picture, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination picture. This function supports parameters governing image quality, compressor type, image depth, custom color tables, and dithering.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FCompressPictureFile

Compresses a single-frame image stored as a picture file and places the result in another picture file, with added control over the compression process.

```
OSErr FCompressPictureFile (
   short srcRefNum,
   short dstRefNum,
   short colorDepth,
   CTabHandle ctable,
   CodecQ quality,
   short doDither,
   short compressAgain,
   ICMProgressProcRecordPtr progressProc,
   CodecType cType,
   CompressorComponent codec
);
```

Parameters

srcRefNum

A file reference number for the source PICT file.

dstRefNum

A file reference number for the destination PICT file. Note that the compressor overwrites the contents of the file referred to by `dstRefNum`. You must open this file with write permissions. The destination file may be the same as the source file specified by the `srcRefNum` parameter.

colorDepth

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 capability structure returned by the `GetCodecInfo` (page 86) function.

ctable

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 `colorDepth` 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 `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` 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 `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

doDither

Indicates whether to dither the image. Use this parameter to indicate whether you want the image to be dithered when it is displayed on a lower-resolution screen. The following constants are available: See these constants:

`defaultDither`

`forceDither`

`suppressDither`

compressAgain

Indicates whether to recompress compressed image data in the `picture`. Use this parameter to control whether any compressed image data that is in the source picture should be decompressed and then recompressed using the current parameters. Set the `value` of this parameter to TRUE to recompress such data. Set the `value` of this parameter to FALSE to leave the data as it is. Note that recompressing the data may have undesirable side effects, including image quality degradation.

progressProc

Points to an `ICMProgressProc` callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress.

cType

A compressor type. You must set this parameter to a valid compressor type constant.

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). See these constants:

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function compresses only image data. Any other types of data in the file, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination picture file. This function supports parameters governing image quality, compressor type, image depth, custom color tables, and dithering.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

FDecompressImage

Decompresses a single-frame image into a pixel map structure, with added control over the decompression process.

```
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,
   ICMDataProcRecordPtr dataProc,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

data

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.

desc

A handle to the `ImageDescription` structure that describes the compressed image.

dst

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

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 `((**desc).width,(**desc).height)`. If you want to decompress the entire source image, set this parameter to `NIL`. If the parameter is `NIL`, the rectangle is set to the rectangle structure of the `ImageDescription` structure.

matrix

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 `matrix` parameter to `NIL`.

mode

The transfer mode for the operation; see `Graphics Transfer Modes`.

mask

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 `NIL`.

matte

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 `NIL`.

matteRect

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 `matte` 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 `srcRect` parameter. If you want to use the entire matte, or if you are not providing a blend matte, set this parameter to `NIL`.

accuracy

A constant (see below) that defines the desired compression accuracy. For a good display of still images, you should specify at least `codecHighQuality`. See these constants:

`codecMinQuality`
`codecLowQuality`
`codecNormalQuality`
`codecHighQuality`
`codecMaxQuality`
`codecLosslessQuality`

codec

A decompressor identifier. Specify a particular decompressor by setting this parameter to its identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance 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. See these constants:

bufferSize

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, set this parameter to 0.

dataProc

Points to an `ICMDataProc` data-loading callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that loads more compressed data. If you have not provided a data-unloading callback, 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 an `ICMProgressProc` progress callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function gives your application greater control over the parameters that guide the decompression operation. If you find that you do not need this level of control, use `DecompressImage` (page 53). Note that this function is invoked through the `StdPix` (page 166) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

JPEG File Interchange Format

qtreadwritejpeg

qtreadwritejpeg.win

VelEng Wavelet

Declared In

`ImageCompression.h`

FindCodec

Determines which of the installed compressors or decompressors has been chosen to field requests made by using one of the special compressor identifiers.

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

Parameters

cType

You must set this parameter to a valid compressor type constant; see `Codec Identifiers`.

`specCodec`

A special codec identifier value (see below). See these constants:

`compressor`

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`

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.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some Image Compression Manager functions allow you to specify a particular compressor component by its identifier. For example, you may use the `codec` parameter to `CompressSequenceBegin` (page 43) to specify a particular compressor to do the compression. The Image Compression Manager also supports several special identifiers (see `specCodec` Constants, above) that allow you to exert some control over the component for a given action without having to know its identifier.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

FixExp2

Undocumented

```
Fixed FixExp2 (
   Fixed src
);
```

Parameters

`src`

A fixed integer.

Return Value

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FixLog2

Undocumented

```
Fixed FixLog2 (
   Fixed src
);
```

Parameters

`src`
A fixed integer.

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FixMulDiv

Undocumented

```
Fixed FixMulDiv (
   Fixed src,
   Fixed mul,
   Fixed divisor
);
```

Parameters

`src`
Undocumented

`mul`
Undocumented

`divisor`
Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FixPow

Undocumented

```
Fixed FixPow (
   Fixed base,
   Fixed exp
);
```

Parameters

`base`
   Undocumented

`exp`
   Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FracSinCos

Undocumented

```
Fract FracSinCos (
   Fixed degree,
   Fract *cosOut
);
```

Parameters

`degree`
   Undocumented

`cosOut`
   Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GDGetScale

Returns the current scale of the given screen graphics device.

```
OSErr GDGetScale (
   GDHandle gdh,
   Fixed *scale,
   short *flags
);
```

Parameters

gdh
: A handle to a screen graphics device.

scale
: Points to a fixed-point field to hold the scale result.

flags
: Points to a short integer that returns the `status` parameter flags for the video driver. Currently, 0 is always returned in this field.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GDHasScale

Returns the closest possible scaling that a particular screen device can be set to in a given pixel depth.

```
OSErr GDHasScale (
   GDHandle gdh,
   short depth,
   Fixed *scale
);
```

Parameters

gdh
: A handle to a screen graphics device.

depth
: The pixel depth of the screen device.

scale

Points to a fixed-point scale value. On input, this field should be set to the desired scale value. On output, this field will contain the closest scale available for the given depth. A scale of 0x10000 indicates normal size, 0x20000 indicates double size, and so on.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns scaling information for a particular screen device for a requested depth. This function allows you to query a screen device without actually changing it. For example, if you specify 0x20000 but the screen device does not support it, `GDHasScale` returns `noErr` and a scale of 0x10000. Because this function checks for a supported depth, your requested depth must be supported by the screen device.

Special Considerations

`GDHasScale` references the video driver through the graphics device structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GDSetScale

Sets a screen graphics device to a new scale.

```
OSErr GDSetScale (
   GDHandle gdh,
   Fixed scale,
   short flags
);
```

Parameters

gdh

A handle to a screen graphics device.

scale

A fixed-point scale value.

flags

Points to a short integer. It returns the `status` parameter flags for the video driver. Currently, 0 is always returned in this field.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetBestDeviceRect

Selects the deepest of all available graphics devices, while treating 16-bit and 32-bit screens as having equal depth.

```
OSErr GetBestDeviceRect (
   GDHandle *gdh,
   Rect *rp
);
```

Parameters

gdh

A pointer to the handle of the rectangle for the chosen device. If you do not need the information in this parameter returned, specify `NIL`.

rp

A pointer to the rectangle that is adjusted for the height of the menu bar if the device is the main device. If you do not need the information in this parameter returned, specify `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function does not center a rectangle on a device. Rather, it returns the rectangle for the best device. The following code sample illustrates its use:

```
// GetBestDeviceRect coding example
// See "Discovering QuickTime," page 265
WindowRef MakeMyWindow (void)
{
    WindowRef   pMacWnd;
    Rect        rectWnd ={0, 0, 120, 160};
    Rect        rectBest;
    // figure out the best monitor for the window
    GetBestDeviceRect(NIL, &rectBest);
    // put the window in the top left corner of that monitor
    OffsetRect(&rectWnd, rectBest.left + 10, rectBest.top + 50);
    // create the window
    pMacWnd =NewCWindow(NIL, &rectWnd, "\pGrabber",
                            TRUE, noGrowDocProc, (WindowRef)-1, TRUE, 0);
    // set the port to the new window
    SetPort(pMacWnd);
    return pMacWnd;
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DigitizerShell

MovieBrowser
Sequence Grabbing
SGDataProcSample

Declared In
`ImageCompression.h`

GetCodecInfo

Returns information about a single compressor component.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetCodecInfo
```

Parameters

`info`
: A pointer to a `CodecInfo` structure. `GetCodecInfo` returns detailed information about the appropriate compressor component in this structure.

`cType`
: Set this parameter to a valid compressor type constant; see `Codec Identifiers`. If you want information about any compressor of the type specified by this parameter, set the `codec` parameter to 0. The Image Compression Manager then returns information about the first compressor it finds of the type you have specified.

`codec`
: Set this parameter to the component identifier of the specific compressor for the request, or to 0 for any compressor. Component identifiers are available in the `CodecNameSpecList` structure returned by `GetCodecNameList` (page 86).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

Declared In
`ImageCompression.h`

GetCodecNameList

Retrieves a list of installed compressor components or types.

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

Parameters

list

A pointer to a field that is to receive a pointer to a `CodecNameSpecList` structure. The Image Compression Manager creates the appropriate list and returns a pointer to that list in the field specified by this parameter.

showAll

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.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The `CodecType` data type defines a field in the `CodecNameSpec` structure that identifies the compression method employed by a given compressor component. See `Codec Identifiers`. 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.

Special Considerations

`GetCodecNameList` creates the `CodecNameSpecList` structure in your application's current heap zone.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetCompressedImageSize

Determines the size, in bytes, of a compressed image.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetCompressedImageSize
```

Parameters

desc

A handle to the `ImageDescription` 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

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, set this parameter to 0.

dataProc

Points to an `ICMDataProc` callback. If the data stream is not all in memory when your program calls `GetCompressedImageSize`, the compressor calls a function you provide that loads more compressed data. If you have not provided a data-loading callback, set this parameter to `NIL`. In this case, the entire image must be in memory at the location specified by the `data` parameter.

dataSize

A pointer to a field that is to receive the size, in bytes, of the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Most applications do not need to use this function because compressed images have a corresponding `ImageDescription` 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.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetCompressedPixMapInfo

Retrieves information about a compressed image.

```
OSErr GetCompressedPixMapInfo (
   PixMapPtr pix,
   ImageDescriptionHandle *desc,
   Ptr *data,
   long *bufferSize,
   ICMDataProcRecord *dataProc,
   ICMProgressProcRecord *progressProc
);
```

Parameters

pix

Points to a structure that holds encoded compressed image data.

desc

A pointer to a field that is to receive a handle to the `ImageDescription` structure that defines the compressed image. If you are not interested in this information, specify `NIL` in this parameter.

data

A pointer to a field that is to receive a pointer to the compressed image data. If the entire compressed image cannot be stored at this location, you can define a data-loading function for this operation. If you are not interested in this information, you may specify `NIL` in this parameter.

bufferSize

A pointer to a field that is to receive the size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If there is no data-loading function defined for this operation, this parameter is ignored. If you are not interested in this information, you may specify `NIL` in this parameter.

dataProc

A pointer to an `ICMDataProc` callback. If there is not enough memory to store the compressed image, the decompressor calls a function you provide that loads more compressed data. If there is no data-loading function for this image, the function sets the `dataProc` field in the function structure to `NIL`. If you are not interested in this information, specify `NIL` in this parameter.

progressProc

A pointer to an `ICMProgressProc` callback. During a decompression operation, the decompressor may occasionally call a function you provide in order to report its progress. If there is no progress function for this image, the function sets the `progressProc` field in the function structure to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function. If you are not interested in progress information, specify `NIL` in this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

DesktopSprites.win

JPEG File Interchange Format

WiredSprites

Declared In

`ImageCompression.h`

GetCompressionTime

Determines the estimated amount of time required to compress a given image.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetCompressionTime
```

Parameters

src

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 `NIL` if you are interested only in information about quality settings.

srcRect

A pointer to a rectangle defining the portion of the source image to compress. You may set this parameter to `NIL` if you are interested only in information about quality settings. `GetCompressionTime` then uses the bounds of the source pixel map.

colorDepth
: 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 `GetCodecInfo` (page 86) function

cType
: You must set this parameter to a valid compressor type constant; see `Codec Identifiers`.

codec
: Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). 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. See these constants:

spatialQuality
: A pointer to a field containing a constant (see below) that defines the desired compressed image quality. The Image Compression Manager sets this field to the closest actual quality that the compressor can achieve. If you are not interested in this information, pass `NIL` in this parameter. See these constants:

    `codecMinQuality`

    `codecLowQuality`

    `codecNormalQuality`

    `codecHighQuality`

    `codecMaxQuality`

    `codecLosslessQuality`

temporalQuality
: A pointer to a field containing a constant (see below) that defines 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. If you are not interested in this information, pass `NIL` in this parameter.

compressTime
: 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.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function 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. 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.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

DigitizerShell

DragAndDrop Shell

MovieGWorlds

QT Internals

Declared In

`ImageCompression.h`

GetCSequenceDataRateParams

Obtains the data rate parameters previously set with SetCSequenceDataRateParams.

```
OSErr GetCSequenceDataRateParams (
   ImageSequence seqID,
   DataRateParamsPtr params
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by `CompressSequenceBegin` (page 43).

params

Points to the `data` rate parameters structure associated with the sequence identifier specified in the `seqID` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetCSequenceFrameNumber

Returns the current frame number of the specified sequence.

```
OSErr GetCSequenceFrameNumber (
   ImageSequence seqID,
   long *frameNumber
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (page 43) function.

frameNumber

A pointer to the current frame number of the sequence identified by the `seqID` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetCSequenceKeyFrameRate

Determines the current key frame rate of a sequence.

```
OSErr GetCSequenceKeyFrameRate (
   ImageSequence seqID,
   long *keyFrameRate
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by `CompressSequenceBegin` (page 43).

keyFrameRate

A pointer to a long integer that specifies the maximum number of frames allowed between key frames. Key frames provide points from which a temporally compressed sequence may be decompressed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetCSequenceMaxCompressionSize

Determines the maximum size an image will be after compression for a given compression sequence.

```
OSErr GetCSequenceMaxCompressionSize (
   ImageSequence seqID,
   PixMapHandle src,
   long *size
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (page 43) function.

src

A handle to the source `PixMap` structure. The compressor uses only the image's size and pixel depth to determine the maximum size of the compressed image.

size

A pointer to a field to receive the maximum size, in bytes, of the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is similar to `GetMaxCompressionSize` (page 104), but operates on a compression sequence instead of requiring the application to pass individual parameters about the source image.

Special Considerations

Before calling `GetCSequenceMaxCompressionSize` you must have already started a compression sequence with `CompressSequenceBegin` (page 43)

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

qteffects.win

qtwiredactions

qtwiredspritesjr.win

Declared In

`ImageCompression.h`

GetCSequencePrevBuffer

Determines the location of the previous image buffer allocated by the compressor.

```
OSErr GetCSequencePrevBuffer (
   ImageSequence seqID,
   GWorldPtr *gworld
);
```

Parameters

seqID
: Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (page 43) function.

gworld
: A pointer to a field to receive a pointer to the `CGrafPort` structure that describes the graphics world for the image buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. If the compressor has not allocated a buffer, `GetCSequencePrevBuffer` returns an error result code.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Note that this function only returns information about buffers that were allocated by the compressor. You cannot use this function to determine the location of a buffer you have provided.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetDSequenceImageBuffer

Determines the location of the offscreen image buffer allocated by a decompressor.

```
OSErr GetDSequenceImageBuffer (
   ImageSequence seqID,
   GWorldPtr *gworld
);
```

Parameters

seqID
: Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

gworld
: A pointer to a field to receive a pointer to the `CGrafPort` structure describing the graphics world for the image buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. It is disposed of for you when the `CDSequenceEnd` (page 25) function is called. If the decompressor has not allocated a buffer, `GetDSequenceImageBuffer` returns an error result code.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetDSequenceMatrix

Gets the matrix that was specified for a decompression sequence by a call to SetDSequenceMatrix, or that was set at DecompressSequenceBegin.

```
OSErr GetDSequenceMatrix (
   ImageSequence seqID,
   MatrixRecordPtr matrix
);
```

Parameters

seqID
  Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 55).

matrix
  Points to a matrix structure that specifies how to transform the image during decompression

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetDSequenceNonScheduledDisplayDirection

Returns the display direction for a decompress sequence.

```
OSErr GetDSequenceNonScheduledDisplayDirection (
   ImageSequence sequence,
   Fixed *rate
);
```

Parameters

sequence
  Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

rate

A pointer to the display direction. Negative values represent backward display and positive values represent forward display.

Return Value

An error code. Returns `noErr` if there is no error.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GetDSequenceNonScheduledDisplayTime

Gets the display time for a decompression sequence.

```
OSErr GetDSequenceNonScheduledDisplayTime (
   ImageSequence sequence,
   TimeValue64 *displayTime,
   TimeScale *displayTimeScale
);
```

Parameters

sequence

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

displayTime

A pointer to a variable to hold the display time.

displayTimeScale

A pointer to a variable to hold the display time scale.

Return Value

An error code. Returns `noErr` if there is no error.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GetDSequenceScreenBuffer

Determines the location of the offscreen screen buffer allocated by a decompressor.

```
OSErr GetDSequenceScreenBuffer (
   ImageSequence seqID,
   GWorldPtr *gworld
);
```

Parameters

seqID

The unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

gworld

A pointer to a field to receive a pointer to the `CGrafPort` structure that describes the graphics world for the screen buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. It is disposed of for you when the `CDSequenceEnd` (page 25) function is called. If the decompressor has not allocated a buffer, `GetDSequenceScreenBuffer` returns an error result code.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetGraphicsImporterForDataRef

Locates and opens a graphics importer component that can be used to draw the image from specified data reference.

```
OSErr GetGraphicsImporterForDataRef (
   Handle dataRef,
   OSType dataRefType,
   ComponentInstance *gi
);
```

Parameters

dataRef

The data reference to be drawn using a graphics importer component.

dataRefType

The type of data reference pointed to by the `dataRef` parameter; see `Data References`. For alias-based data references, the `dataRef` handle contains an `AliasRecord` and `dataRefType` is set to `rAliasType`.

gi

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found, this parameter will be set to `NIL`. If `GetGraphicsImporterForDataRef` is able to locate a graphics importer for the data reference, the returned graphics importer `ComponentInstance` will already be set up to draw from the specified data reference to the current port.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function tries to locate a graphics importer component for the specified data reference by checking the file extension (such as .GIF or .JPG), the Macintosh file type, and the MIME type of the file. The file extension is retrieved from the data reference by using `DataHGetFileName` to call the data handler associated with the data reference. If a graphics importer cannot be found using the file's type, file extension, or MIME type, `GetGraphicsImporterForDataRef` asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming; to bypass it, call `GetGraphicsImporterForDataRefWithFlags` (page 98) instead.

Special Considerations

The caller of `GetGraphicsImporterForDataRef` is responsible for closing the returned `ComponentInstance` using `CloseComponent`. You must call `CloseComponent` when you are finished with the importer.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ComboBoxPrefs

Graphic Import-Export

TextNameTool

ThreadsExporter

ThreadsImporter

Declared In

`ImageCompression.h`

GetGraphicsImporterForDataRefWithFlags

Locates and opens a graphics importer component for a data reference with flags that control the search process.

```
OSErr GetGraphicsImporterForDataRefWithFlags (
   Handle dataRef,
   OSType dataRefType,
   ComponentInstance *gi,
   long flags
);
```

Parameters

`dataRef`

The data reference to be drawn using a graphics importer component.

`dataRefType`

The type of data reference pointed to by the `dataRef` parameter; see `Data References`. For alias-based data references, the `dataRef` handle contains an `AliasRecord` and `dataRefType` is set to `rAliasType`.

gi

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found, this parameter will be set to `NIL`. If `GetGraphicsImporterForDataRefWithFlags` is able to locate a graphics importer for the data reference, the returned graphics importer `ComponentInstance` will already be set up to draw from the specified data reference to the current port.

flags

Contains flags (see below) that control the graphics importer search process. See these constants:
`kDontUseValidateToFindGraphicsImporter`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function tries to locate a graphics importer component for the specified data reference by checking the file extension (such as .GIF or .JPG), the Macintosh file type, and the MIME type of the file. The file extension is retrieved from the data reference by using `DataHGetFileName` to call the data handler associated with the data reference. If a graphics importer cannot be found using the file's type, file extension, or MIME type, this function asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming; to bypass it, pass `kDontUseValidateToFindGraphicsImporter` in the `flags` parameter.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetGraphicsImporterForFile

Locates and opens a graphics importer component that can be used to draw a specified file.

```
OSErr GetGraphicsImporterForFile (
   const FSSpec *theFile,
   ComponentInstance *gi
);
```

Parameters

theFile

The file to be drawn using a graphics importer component.

gi

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found for the specified file, the `gi` will be set to `NIL`. If `GetGraphicsImporterForFile` is able to locate a graphics importer for the file, the returned graphics importer `ComponentInstance` will already be set up to draw the specified file to the current port.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function first tries to locate a graphics importer component for the specified file based on its file type. If it is unable to locate a graphics importer component based on the Macintosh file type, or the call is made on a non-Macintosh file, `GetGraphicsImporterForFile` will try to locate a graphics importer component based on the file extension (such as `.JPG` or `.GIF`). If a graphics importer cannot be found using the file's type or extension, `GetGraphicsImporterForFile` asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming. To bypass the validation attempt, call `GetGraphicsImporterForFileWithFlags` (page 100) instead. The following code sample illustrates the use of `GetGraphicsImporterForFile`:

```
// Get a graphics importer for the image file, determine the natural size
// of the image, and draw the image
// See "Discovering QuickTime," page 274
void drawFile(const FSSpec *fss, const Rect *boundsRect)
    {
        GraphicsImportComponent gi;
        GetGraphicsImporterForFile(fss, &gi);
        GraphicsImportSetBoundsRect(gi, boundsRect);
        GraphicsImportDraw(gi);
        CloseComponent(gi);
    }
```

Special Considerations

The caller of `GetGraphicsImporterForFile` is responsible for closing the returned `ComponentInstance` using `CloseComponent`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtgraphics.win

qtstreamsplicer.win

vrmakepano

Declared In

`ImageCompression.h`

GetGraphicsImporterForFileWithFlags

Locates and opens a graphics importer component for a file with flags that control the search process.

```
OSErr GetGraphicsImporterForFileWithFlags (
   const FSSpec *theFile,
   ComponentInstance *gi,
   long flags
);
```

Parameters

theFile

The file to be drawn using a graphics importer component.

gi

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found for the specified file, the `gi` will be set to `NIL`. If `GetGraphicsImporterForFileWithFlags` is able to locate a graphics importer for the file, the returned graphics importer `ComponentInstance` will already be set up to draw the specified file to the current port.

flags

Contains flags (see below) that control the graphics importer search process. See these constants:

`kDontUseValidateToFindGraphicsImporter`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function first tries to locate a graphics importer component for the specified file based on its file type. If it is unable to locate a graphics importer component based on the Macintosh file type, or the call is made on a non-Macintosh file, `GetGraphicsImporterForFile` will try to locate a graphics importer component based on the file extension (such as .JPG or .GIF). If a graphics importer cannot be found using the file's type or extension, `GetGraphicsImporterForFile` asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming. To bypass the validation attempt, pass `kDontUseValidateToFindGraphicsImporter` in the `flags` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetImageDescriptionCTable

Gets the custom color table for an image.

```
OSErr GetImageDescriptionCTable (
   ImageDescriptionHandle desc,
   CTabHandle *ctable
);
```

Parameters

desc

A handle to the appropriate `ImageDescription` structure.

ctable

A pointer to a field that is to receive a color table handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns the color table for the image described by the `ImageDescription` structure that is referred to by the `desc` parameter. The function correctly sizes the handle for the color table it returns.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

Declared In

`ImageCompression.h`

GetImageDescriptionExtension

Returns a new handle with the data from a specified image description extension.

```
OSErr GetImageDescriptionExtension (
   ImageDescriptionHandle desc,
   Handle *extension,
   long idType,
   long index
);
```

Parameters

desc

A handle to the appropriate `ImageDescription` structure.

extension

A pointer to a field to receive a handle to the returned data. The `GetImageDescriptionExtension` function returns the extended data for the image described by the `ImageDescription` structure referred to by the `desc` parameter. The function correctly sizes the handle for the data it returns.

idType

Specifies the extension's type value. Use this parameter to determine the `data` type of the `extension`. This parameter contains a four-character code, similar to an `OSType` field value.

index

The index of the extension to retrieve. This is a number between 1 and the count returned by `CountImageDescriptionExtensionType` (page 52).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows the application to get a copy of a specified image description extension. Note that each compressor type may have its own format for the extended data that is stored with an image. The extended data is similar in concept to the user data that applications can associate with QuickTime movies. Once you have added extended data to an image, you cannot delete it.

Special Considerations

The Image Compression Manager allocates a new handle and passes it back in the `extension` parameter. Your application should dispose of the handle when it is no longer needed.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ImproveYourImage

Declared In
`ImageCompression.h`

GetMatrixType

Obtains information about a matrix.

```
short GetMatrixType (
   const MatrixRecord *m
);
```

Parameters

m

Points to the `MatrixRecord` structure for this operation.

Return Value
A constant (see below) that defines the type of the matrix.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

Graphic Import-Export

ImproveYourImage

Declared In
`ImageCompression.h`

GetMaxCompressionSize

Determines the maximum size an image will be after compression.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetMaxCompressionSize
```

Parameters

src

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.

srcRect

A pointer to a rectangle defining the portion of the source image that is to be compressed. You may set this parameter to `NIL` if you are interested only in information about quality settings. `GetCompressionTime` (page 89) then uses the bounds of the source pixel map.

colorDepth

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 `GetCodecInfo` (page 86).

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

You must set this parameter to a valid compressor type constant; see `Codec Identifiers`.

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). 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. See these constants:

size

A pointer to a field to receive the size, in bytes, of the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function 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. The following code sample illustrates its use:

```
// GetMaxCompressionSize coding example
// See "Discovering QuickTime," page 286
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                    lMaxCompressedSize =0;
    Handle                  hCompressedData =NIL;
    Ptr                     pCompressedData;
    ImageDescriptionHandle  hImageDesc =NIL;
    OSErr                   nErr;
    PicHandle               hpicPicture =NIL;
    Rect                    rectImage =(**hpmImage).bounds;
    CodecType               dwCodecType =kJPEGCodecType;
    CodecComponent          codec =(CodecComponent)anyCodec;
    CodecQ                  dwSpatialQuality =codecNormalQuality;
    short                   nDepth =0;          // let ICM choose depth
    nErr =GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                                    dwSpatialQuality,
                                    dwCodecType,
                                    (CompressorComponent)codec,
                                    &lMaxCompressedSize);
    if (nErr !=noErr)
        return NIL;

    hImageDesc =(ImageDescriptionHandle)NewHandle(4);
    hCompressedData =NewHandle(lMaxCompressedSize);
    if ((hCompressedData !=NIL) && (hImageDesc !=NIL)) {
        MoveHHi(hCompressedData);
        HLock(hCompressedData);
        pCompressedData =StripAddress(*hCompressedData);

        nErr =CompressImage(hpmImage,
                                    &rectImage,
                                    dwSpatialQuality,
                                    dwCodecType,
                                    hImageDesc,
                                    pCompressedData);

        if (nErr ==noErr) {
            ClipRect(&rectImage);
            hpicPicture =OpenPicture(&rectImage);
            nErr =DecompressImage(pCompressedData,
                                    hImageDesc,
                                    hpmImage,
                                    &rectImage,
                                    &rectImage,
                                    srcCopy,
                                    NIL);
            ClosePicture();
        }
        if (theErr || (GetHandleSize((Handle)hpicPicture) ==
                                        sizeof(Picture))) {
            KillPicture(hpicPicture);
            hpicPicture =NIL;
        }
    }
    if (hImageDesc !=NIL)
        DisposeHandle((Handle)hImageDesc);
    if (hCompressedData !=NIL)
        DisposeHandle(hCompressedData);
    return hpicPicture;
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

Fiendishthngs

Inside Mac ICM Code

vrmakepano

VRMakePano Library

Declared In
`ImageCompression.h`

GetNextImageDescriptionExtensionType

Retrieves an image description structure extension type.

```
OSErr GetNextImageDescriptionExtensionType (
   ImageDescriptionHandle desc,
   long *idType
);
```

Parameters

desc

A handle to an `ImageDescription` structure.

idType

A pointer to an integer that indicates the type of the extension after which this function is to return the `next` extension type. `Point` to a value of 0 to return the first type found.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows your application to search for all the types of extensions in an `ImageDescription` structure. The `idType` parameter should be set to 0 to start the search. When no more extension types can be found, the function will set this field to 0.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetPictureFileHeader

Extracts the picture frame and file header from a specified picture file.

```
OSErr GetPictureFileHeader (
   short refNum,
   Rect *frame,
   OpenCPicParams *header
);
```

Parameters

refNum

A file reference number for the source PICT file.

frame

A pointer to a rectangle that is to receive the picture frame rectangle of the picture file. This function places the `picFrame` rectangle from the picture structure into the rectangle referred to by the `frame` parameter. If you are not interested in this information, pass `NIL` in this parameter.

header

A pointer to an `OpenCPicParams` structure. The `GetPictureFileHeader` function places the header from the specified picture file into this structure. If you are not interested in this information, pass `NIL` in this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your program can use the information returned in the `header` parameter to determine how to draw an image without having to read the picture file.

Special Considerations

Note that this function always returns a version 2 header. If the source file is a version 1 PICT file, the `GetPictureFileHeader` function converts the header into version 2 format before returning it to your application. See *Inside Macintosh: Imaging With QuickDraw* for more information about picture headers.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DrawTextCodec

Declared In

`ImageCompression.h`

GetSimilarity

Compares a compressed image to a picture stored in a pixel map and returns a value indicating the relative similarity of the two images.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetSimilarity
```

Parameters

src

A handle to the noncompressed image. The image must be stored in a pixel map structure.

srcRect

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 `ImageDescription` structure specified by the `desc` parameter.

desc

A handle to the `ImageDescription` 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.

similarity

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).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

HitTestDSequenceData

Undocumented

```
OSErr HitTestDSequenceData (
   ImageSequence seqID,
   void *data,
   Size dataSize,
   Point where,
   long *hit,
   long hitFlags
);
```

Parameters

seqID

The unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

data

A pointer to data.

dataSize

The size of the data.

`where`

A `Point` structure that defines the hit location.

`hit`

Undocumented

`hitFlags`

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMDecompressComplete

Signals the completion of a decompression operation.

```
void ICMDecompressComplete (
   ImageSequence seqID,
   OSErr err,
   short flag,
   ICMCompletionProcRecordPtr completionRtn
);
```

Parameters

`seqID`

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

`err`

Indicates whether the operation succeeded or failed. Set this parameter to 0 for successful operations. For failed operations, set the error code appropriate for the failure. For canceled operations (for example, when the ICM calls your component's `ImageCodecFlush` function), set this parameter to -1.

`flag`

Completion flags (see below). Note that you may set more than one of these flags to 1. See these constants:

`codecCompletionSource`

`codecCompletionDest`

`codecCompletionDontUnshield`

`completionRtn`

A pointer to an `ICMCompletionProcRecord` structure. That structure identifies the application's completion function and contains a reference constant associated with the frame. Your component obtains this structure as part of the `CodecDecompressParams` structure provided by the Image Compression Manager at the start of the decompression operation.

Discussion
Your component must call this function at the end of decompression operations.

Special Considerations

Prior to QuickTime 2.0, decompressor components called the application's completion function directly. For compatibility, that method is still supported except for scheduled asynchronous decompression operations, which must use the `ICMDecompressComplete` call. Newer decompressors should always use `ICMDecompressComplete` rather than calling the completion function directly, regardless of the type of decompression operation.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMDecompressCompleteS

Undocumented

```
OSErr ICMDecompressCompleteS (
   ImageSequence seqID,
   OSErr err,
   short flag,
   ICMCompletionProcRecordPtr completionRtn
);
```

Parameters

`seqID`
The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

`err`
Indicates whether the operation succeeded or failed. See `Error Codes`.

`flag`
Undocumented

`completionRtn`
A pointer to an `ICMCompletionProcRecord` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMGetPixelFormatInfo

Retrieves pixel format information.

```
OSErr ICMGetPixelFormatInfo (
   OSType PixelFormat,
   ICMPixelFormatInfoPtr theInfo
);
```

Parameters

PixelFormat

A constant that identifies the format; see `Pixel Formats`.

theInfo

A pointer to your `ICMPixelFormatInfo` structure in which information is returned. You should initialize the `size` field of this structure with `sizeof (ICMPixelFormatInfo)`. The function will not copy more than this number of bytes into the structure. On return, the `size` field contains the actual size of the data structure. If this amount is greater the size you passed in, that means you didn't retrieve all of the information.

Return Value

Returns `cDepthErr` if the pixel format is not valid. For other errors, see `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMSequenceGetChainMember

Undocumented

```
OSErr ICMSequenceGetChainMember (
   ImageSequence seqID,
   ImageSequence *retSeqID,
   long flags
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

retSeqID

Undocumented

flags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMSequenceGetInfo

Gets multiprocessing properties for compression and decompression sequences.

```
OSErr ICMSequenceGetInfo (
   ImageSequence seqID,
   OSType which,
   void *data
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

which

A constant (see below) that determines the property to be returned. See these constants:

`kICMSequenceTaskWeight`

`kICMSequenceTaskName`

data

The value of the property indicated by the which parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function determines if ICM clients have requested that multiprocessor tasks assisting compression and decompression operations use specific task weights and task names.

Special Considerations

Apple's multiprocessing capability supports both co-operatively scheduled tasks and preemptively scheduled tasks. The support for preemptively tasks allow applications to create symmetrically scheduled preemptive tasks that can be run on a single processor machine, and will take full advantage of multiple processors when they are installed.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMSequenceLockBits

Undocumented

```
OSErr ICMSequenceLockBits (
   ImageSequence seqID,
   PixMapPtr dst,
   long flags
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

dst

A pointer to a `PixMap` structure.

flags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMSequenceSetInfo

Sets multiprocessing properties for compression and decompression sequences.

```
OSErr ICMSequenceSetInfo (
   ImageSequence seqID,
   OSType which,
   void *data,
   Size dataSize
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

which

A constant (see below) that determines the property to be set. See these constants:

`kICMSequenceTaskWeight`

`kICMSequenceTaskName`

data

The value of the property to be set.

dataSize

The length in bytes of the `data` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function lets ICM clients request that multiprocessor tasks assisting compression and decompression operations use specific task weights and task names.

Special Considerations

Apple's multiprocessing capability supports both co-operatively scheduled tasks and preemptively scheduled tasks. The support for preemptively tasks allow applications to create symmetrically scheduled preemptive tasks that can be run on a single processor machine, and will take full advantage of multiple processors when they are installed.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMSequenceUnlockBits

Undocumented

```
OSErr ICMSequenceUnlockBits (
   ImageSequence seqID,
   long flags
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

flags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMSetPixelFormatInfo

Lets you define your own pixel format.

```
OSErr ICMSetPixelFormatInfo (
   OSType PixelFormat,
   ICMPixelFormatInfoPtr theInfo
);
```

Parameters

PixelFormat

A pixel format constant. See `Pixel Formats`.

theInfo

A pointer to an `ICMPixelFormatInfo` structure containing a definition of the new pixel format.

Return Value

Returns `paramErr` if the format is already defined. See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

OpenGLCompositorLab

SoftVideoOutputComponent

Declared In

`ImageCompression.h`

ICMShieldSequenceCursor

Hides the cursor during decompression operations.

```
OSErr ICMShieldSequenceCursor (
   ImageSequence seqID
);
```

Parameters

seqID

The unique sequence identifier, assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55), for which to shield the cursor.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

For correct image display behavior, the cursor must be shielded (hidden) during decompression. By default, the Image Compression Manager handles the cursor for you, hiding it at the beginning of a decompression operation and revealing it at the end. With scheduled asynchronous decompression, however, the ICM cannot do as precise a job of managing the cursor, because it does not know exactly when scheduled operations actually begin and end. While the ICM can still manage the cursor, it must hide the cursor when each request is queued, rather than when the request is serviced. This may result in the cursor remaining hidden for long periods of time. To achieve better cursor behavior, you can choose to manage the cursor in your decompressor component. If you so choose, you can use this function to hide the cursor; the ICM displays the cursor when you call `ICMDecompressComplete` (page 109). In this manner, the cursor is hidden only when your component is decompressing and displaying the frame.

Special Considerations

This function may be called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ImageFieldSequenceBegin

Initiates an image field sequence operation and specifies the input and output data format.

```
OSErr ImageFieldSequenceBegin (
   ImageFieldSequence *ifs,
   ImageDescriptionHandle desc1,
   ImageDescriptionHandle desc2,
   ImageDescriptionHandle descOut
);
```

Parameters

ifs

On return, contains the unique sequence identifier assigned to the sequence.

desc1

An `ImageDescription` structure describing the format and characteristics of the data to be passed to `ImageFieldSequenceExtractCombine` (page 117) through the data1 parameter.

desc2

An `ImageDescription` structure describing the format and characteristics of the data to be passed to the `ImageFieldSequenceExtractCombine` function through the data2 parameter. Set to `NIL` if the requested operation uses only one input frame.

descOut

The desired format of the resulting frames. Typically this is the same format specified by the desc1 and desc2 parameters.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to set up an image field sequence operation and specify the input and output data format.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageFieldSequenceEnd

Ends an image field sequence operation.

```
OSErr ImageFieldSequenceEnd (
   ImageFieldSequence ifs
);
```

Parameters

`ifs`
: The unique sequence identifier that was returned by the `ImageFieldSequenceBegin` (page 116) function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You must call this function to terminate an image field sequence operation.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageFieldSequenceExtractCombine

Performs field operations on video data.

```
OSErr ImageFieldSequenceExtractCombine (
   ImageFieldSequence ifs,
   long fieldFlags,
   void *data1,
   long dataSize1,
   void *data2,
   long dataSize2,
   void *outputData,
   long *outDataSize
);
```

Parameters

`ifs`
: The unique sequence identifier that was returned by `ImageFieldSequenceBegin` (page 116).

`fieldFlags`

Flags (see below) that specify the operation to be performed. A correctly formed request will specify two input fields, mapping one to the odd output field and the other to the even output field. See these constants:

`evenField1ToEvenFieldOut`

`evenField1ToOddFieldOut`

`oddField1ToEvenFieldOut`

`oddField1ToOddFieldOut`

`evenField2ToEvenFieldOut`

`evenField2ToOddFieldOut`

`oddField2ToEvenFieldOut`

`oddField2ToOddFieldOut`

`data1`

A pointer to a buffer containing the `data` of input field one.

`dataSize1`

The size of the data1 buffer.

`data2`

A pointer to a buffer containing the `data` of input field two. Set to `NIL` if the requested operation uses only one input frame.

`dataSize2`

The size of the data2 buffer. Set to 0 if the requested operation uses only one input frame.

`outputData`

A pointer to a buffer to receive the resulting frame. Use `GetMaxCompressionSize` (page 104) to determine the amount of memory to allocate for this buffer.

`outDataSize`

On output this parameter returns the actual size of the data.

Return Value

Returns the `codecUnimpErr` result code if there is no codec present in the system that can perform the requested operation. See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function provides a method for working directly with fields of interlaced video. You can use it to change the field dominance of an image by reversing the two fields, or to create or remove the effects of the 3:2 pulldown commonly performed when transferring film to NTSC videotape. Because this function operates directly on the compressed video data, it is faster than working with decompressed images. It also has the added benefit of eliminating any image quality degradation that might result from lossy codecs.

This function accepts one or two compressed images as input and creates a single compressed image on output. You specify the operation to be performed using the `fieldFlags` parameter.

Special Considerations

The Apple Component Video (YUV) and Motion JPEG codecs currently support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscodeDisposeFrameData

Disposes transcoded image data.

```
OSErr ImageTranscodeDisposeFrameData (
   ImageTranscodeSequence its,
   void *dstData
);
```

Parameters
`its`
: The image transcoder sequence that was used to generate the transcoded data.

`dstData`
: A pointer to the transcoded image data generated by the `ImageTranscodeFrame` (page 119) function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When the transcoded image data returned by `ImageTranscodeFrame` (page 119) is no longer needed, use this function to dispose of the data. Only the image transcoder that generated the data can properly dispose of it.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscodeFrame

Transcodes a frame of image data.

```
OSErr ImageTranscodeFrame (
   ImageTranscodeSequence its,
   void *srcData,
   long srcDataSize,
   void **dstData,
   long *dstDataSize
);
```

Parameters
`its`
: The image transcoder sequence to use to perform the transcoding operation.

`srcData`
: A pointer to the source data to transcode.

srcDataSize
: The size of the compressed source image data in bytes.

dstData
: On return, a pointer to the transcoded image data.

dstDataSize
: On return, the size of the transcoded image data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

After creating the image transcoder sequence, using `ImageTranscodeSequenceBegin` (page 120), use this function to transcode a frame of image data. The caller is responsible for disposing of the transcoded data using `ImageTranscodeDisposeFrameData` (page 119).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ImageTranscodeSequenceBegin

Initiates an image transcoder sequence operation.

```
OSErr ImageTranscodeSequenceBegin (
   ImageTranscodeSequence *its,
   ImageDescriptionHandle srcDesc,
   OSType destType,
   ImageDescriptionHandle *dstDesc,
   void *data,
   long dataSize
);
```

Parameters

its
: The image transcoder sequence identifier. If the operation fails, the value pointed to is set to `NIL`.

srcDesc
: The `ImageDescription` structure for the source compressed image data.

destType
: The desired compression format into which to transcode the source data.

dstDesc
: On return, an `ImageDescription` structure for the data which will be generated by the image transcoding sequence.

data
: A pointer to first frame of compressed data to transcode. Set to `NIL` of not available.

dataSize
: The size of the compressed data, in bytes. Set to 0 if no data is provided.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If no transcoder is available to perform the requested transcoding operation, a `cantFindHandler` error is returned.

Discussion
This function begins an image transcoder sequence operation and returns the sequence identifier in the `its` parameter. The caller is responsible for disposing of the `ImageDescription` structure that is returned in the `dstDesc` parameter.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscodeSequenceEnd

Ends an image transcoder sequence operation.

```
OSErr ImageTranscodeSequenceEnd (
   ImageTranscodeSequence its
);
```

Parameters

`its`
: The identifier of the image transcoder sequence to dispose. It is safe to pass a value of 0 in this parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You must call this function to terminate an image transcoder sequence operation and dispose of the sequence.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

InverseMatrix

Creates a new matrix that is the inverse of a specified matrix.

```
Boolean InverseMatrix (
   const MatrixRecord *m,
   MatrixRecord *im
);
```

Parameters

m

A pointer to the source `MatrixRecord` structure for the operation.

im

A pointer to a `MatrixRecord` structure that is to receive the new matrix. The function updates this structure so that it contains a matrix that is the inverse of that specified by the `m` parameter.

Return Value

A Boolean value of TRUE if `InverseMatrix` was able to create an inverse matrix, FALSE otherwise.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtactiontargets

qtactiontargets.win

qtwiredspritesjr

qtwiredspritesjr.win

Declared In

`ImageCompression.h`

MakeFilePreview

Creates a preview for a file.

```
OSErr MakeFilePreview (
   short resRefNum,
   ICMProgressProcRecordPtr progress
);
```

Parameters

resRefNum

The resource file for this operation. You must have opened this resource file with write permission. If there is a preview in the specified file, the Movie Toolbox replaces that preview with a new one.

progress

A pointer to an `ICMProgressProcRecord` structure. During the process of creating the preview, the Movie Toolbox may occasionally call a function you provide in order to report its progress. You can then use this information to keep the user informed.

Set this parameter to -1 to use the default progress function. If you specify a progress function, it must comply with the interface defined for Image Compression Manager progress functions; see "Image Compression Manager" in *Inside Macintosh: QuickTime* for more information. Set this parameter to `NIL` to prevent the Movie Toolbox from calling a progress function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You should create a preview whenever you save a movie. You specify the file by supplying a reference to its resource file. You must have opened this resource file with write permission. If there is a preview in the specified file, the Movie Toolbox replaces that preview with a new one.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo

qtinfo.win

Declared In
`ImageCompression.h`

MakeImageDescriptionForEffect

Returns an ImageDescription structure you can use to help create a sample description for an effect.

```
OSErr MakeImageDescriptionForEffect (
   OSType effectType,
   ImageDescriptionHandle *idh
);
```

Parameters

effectType
: The four-character code identifying the type of effect to make an image description for. See `Effects Codes`.

idh
: The handle of an `ImageDescription` structure. On entry, this parameter normally points to an `ImageDescription` structure whose contents are `NIL`. On return, the structure is correctly filled out for the selected effect type.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
To create a sample description, you create and fill out a data structure of type `ImageDescription`. This function simplifies this process. Only sample descriptions made with this function can be used in stacked effects, where one effect track acts as a source for another.

The following sample code creates a sample description:.

```
// MakeImageDescriptionForEffect coding example
// Return a new image description with default and specified values.
ImageDescriptionHandle QTEffSeg_MakeSampleDescription (
                   OSType theEffectType, short theWidth, short theHeight)
{
    ImageDescriptionHandle      mySampleDesc =NIL;
#if USES_MAKE_IMAGE_DESC_FOR_EFFECT
    OSErr  myErr =noErr;
    // create a new sample description
    myErr =MakeImageDescriptionForEffect(theEffectType, &mySampleDesc);
    if (myErr !=noErr)
        return(NIL);
#else
    // create a new sample description
    mySampleDesc =(ImageDescriptionHandle)
                                NewHandleClear(sizeof(ImageDescription));
    if (mySampleDesc ==NIL)
        return(NIL);

    // fill in the fields of the sample description
    (**mySampleDesc).cType =theEffectType;
    (**mySampleDesc).idSize =sizeof(ImageDescription);
    (**mySampleDesc).hRes =72L << 16;
    (**mySampleDesc).vRes =72L << 16;
    (**mySampleDesc).frameCount =1;
    (**mySampleDesc).depth =0;
    (**mySampleDesc).clutID =-1;
#endif

    (**mySampleDesc).vendor =kAppleManufacturer;
    (**mySampleDesc).temporalQuality =codecNormalQuality;
    (**mySampleDesc).spatialQuality =codecNormalQuality;
    (**mySampleDesc).width =theWidth;
    (**mySampleDesc).height =theHeight;

    return(mySampleDesc);
}
```

Version Notes
Introduced in QuickTime 4. Image descriptions built using sample code from earlier versions of QuickTime cannot be used when stacking effects.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtaddeffectseg.win

qteffects.win

qtshoweffect.win

samplemakeeffectmovie

samplemakeeffectmovie.win

Declared In
`ImageCompression.h`

MakeImageDescriptionForPixMap

Fills out an ImageDescription structure corresponding to a PixMap structure.

```
OSErr MakeImageDescriptionForPixMap (
   PixMapHandle pixmap,
   ImageDescriptionHandle *idh
);
```

Parameters

pixmap

A handle to a `PixMap` structure.

idh

The handle of an `ImageDescription` structure. On entry, this parameter normally points to an `ImageDescription` structure whose contents are `NIL`. On return, the structure is correctly filled out for the selected `PixMap`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ImproveYourImage

SGDataProcSample

VideoProcessing

vrscript

vrscript.win

Declared In

`ImageCompression.h`

MakeThumbnailFromPicture

Creates a thumbnail picture from a specified Picture structure.

```
OSErr MakeThumbnailFromPicture (
   PicHandle picture,
   short colorDepth,
   PicHandle thumbnail,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

picture

A handle to the image from which the thumbnail is to be extracted. The image must be stored in a `Picture` structure.

colorDepth

The depth at which the image is likely to be viewed. 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.

thumbnail

A handle to the destination `Picture` structure for the thumbnail image. The compressor resizes this handle for the resulting data.

progressProc

A pointer to an `ICMProgressProcRecord` structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure. 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.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtinfo

qtinfo.win

Declared In

`ImageCompression.h`

MakeThumbnailFromPictureFile

Creates a thumbnail picture from a specified picture file.

```
OSErr MakeThumbnailFromPictureFile (
   short refNum,
   short colorDepth,
   PicHandle thumbnail,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

refNum

A file reference number for the PICT file from which the thumbnail is to be extracted.

colorDepth

The depth at which the image is likely to be viewed. 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.

thumbnail

A handle to the destination picture structure for the thumbnail image. The compressor resizes this handle for the resulting data.

progressProc

A pointer to an `ICMProgressProcRecord` structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

MakeThumbnailFromPixMap

Creates a thumbnail picture from a specified PixMap structure.

```
OSErr MakeThumbnailFromPixMap (
   PixMapHandle src,
   const Rect *srcRect,
   short colorDepth,
   PicHandle thumbnail,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

`src`
A handle to the image from which the thumbnail is to be extracted. The image must be stored in a `PixMap` structure.

`srcRect`
A pointer to a `Rect` structure that defines the portion of the image to use for the thumbnail.

`colorDepth`
The depth at which the image is likely to be viewed. 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.

`thumbnail`
A handle to the destination picture structure for the thumbnail image. The compressor resizes this handle for the resulting data.

`progressProc`
A pointer to an `ICMProgressProcRecord` structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

MapMatrix

Alters an existing matrix so that it defines a transformation from one rectangle to another.

```
void MapMatrix (
   MatrixRecord *matrix,
   const Rect *fromRect,
   const Rect *toRect
);
```

Parameters

`matrix`
: A pointer to a matrix structure. The `MapMatrix` function modifies this matrix so that it performs a transformation in the rectangle specified by the `toRect` parameter that is analogous to the transformation it currently performs in the rectangle specified by the `fromRect` parameter.

`fromRect`
: A pointer to the source `Rect` structure.

`toRect`
: A pointer to the destination `Rect` structure.

Discussion
`MapMatrix` affects only the scaling and translation attributes of the matrix.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In
`ImageCompression.h`

NewImageGWorld

Creates an offscreen graphics world.

```
ComponentResult ADD_IMAGECODEC_BASENAME() NewImageGWorld
```

Parameters

`gworld`
: A pointer to a graphic world created using the width, height, depth, and color table specified in the `ImageDescription` structure pointed to in the `idh` parameter.

idh

A handle to an `ImageDescription` structure that contains information for the graphics world pointed to by the `gworld` parameter.

flags

Graphics world creation flags (see below). The `pixPurge`, `noNewDevice`, `useTempMem`, `keepLocal`, `pixelsPurgeable`, and `pixelsLocked` flags are commands to this function; the others are returned by this function. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

PtInDSequenceData

Tests to see if a compressed image contains data at a a given point.

```
OSErr PtInDSequenceData (
   ImageSequence seqID,
   void *data,
   Size dataSize,
   Point where,
   Boolean *hit
);
```

Parameters

seqID

The unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

data

Pointer to compressed data in the format specified by the `desc param`.

dataSize

`Size` of the compressed data referred to by the data `param`.

where

A QuickDraw `Point` structure of value (0,0), based at the top-left corner of the image.

hit

A pointer to a field to receive the Boolean indicating whether or not the image contained data at the specified point. The Boolean will be set to TRUE if the point specified by the `where` parameter is contained within the compressed image data specified by the data `param`, or FALSE if the specified point falls within a blank portion of the image.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

`PtInDSequenceData` allows the application to perform hit testing on compressed data.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTGetFileNameExtension

Gets the extension to a file name.

```
OSErr QTGetFileNameExtension (
   ConstStrFileNameParam fileName,
   OSType fileType,
   OSType *extension
);
```

Parameters

`fileName`

A file name string.

`fileType`

A file type; see `File Types and Creators`.

`extension`

A pointer to the file name extension string.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtdataexchange

qtdataexchange.win

Declared In

`ImageCompression.h`

QTGetPixelFormatDepthForImageDescription

For a given pixel format, returns the depth value that should be used in image descriptions.

```
short QTGetPixelFormatDepthForImageDescription (
   OSType PixelFormat
);
```

Parameters

PixelFormat

The image description's pixel format; see `Pixel Formats`.

Return Value

The pixel depth for that format.

Discussion

Given a pixel format, this function returns the corresponding depth value that should be used in image descriptions. Such a value is not the literal number of bits per pixel, but the closest corresponding classic QuickDraw depth. For any pixel format with an alpha channel, it is 32. For grayscale pixel formats of 8 or more bits per pixel, it is 40. For color quantized to 5 or 6 bits per component, it is 16. For all other color pixel formats, it is 24.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ImageCompression.h`

QTGetPixelSize

Returns the bits per pixel for a given pixel format.

```
short QTGetPixelSize (
   OSType PixelFormat
);
```

Parameters

PixelFormat

A constant that identifies the pixel format; see `Pixel Formats`. This function returns meaningful information only for non-planar formats.

Return Value

The bits per pixel. Returns 0 if the format is unknown.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dimmer2Effect

Dimmer2Effect.win

ElectricImageComponent

ElectricImageComponent.win

GreyscaleEffectSample

Declared In
`ImageCompression.h`

QTGetPixMapHandleGammaLevel

Retrieves the current PixMap extension's gamma level setting.

```
Fixed QTGetPixMapHandleGammaLevel (
   PixMapHandle pm
);
```

Parameters

pm

A handle to a `PixMap` structure that has a `PixMapExtension` structure.

Return Value
On return, the gamma level previously set (or the default level) for the pixel map referenced by the `pm` parameter.

Discussion
A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTGetPixMapHandleRequestedGammaLevel

Retrieves the current PixMap extension's requested gamma level setting.

```
Fixed QTGetPixMapHandleRequestedGammaLevel (
   PixMapHandle pm
);
```

Parameters

pm

A handle to a `PixMap` structure that has a `PixMapExtension` structure.

Return Value
On return, the requested gamma level previously set (or the default level) for the pixel map referenced by the `pm` parameter.

Discussion
A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure. The requested gamma level is used to control what gamma conversion is attempted during decompression. The requested gamma level may differ from the actual gamma level depending on the compressed data and the capabilities of the codecs involved.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTGetPixMapHandleRowBytes

Gets the rowBytes value for a pixel map accessed by a handle.

```
long QTGetPixMapHandleRowBytes (
   PixMapHandle pm
);
```

Parameters

pm

A handle to a `PixMap` structure.

Return Value
The `rowBytes` value.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

qteffects.win

qtsprites.win

qtwiredactions

qtwiredspritesjr.win

Declared In
`ImageCompression.h`

QTGetPixMapPtrGammaLevel

Retrieves the current PixMap extension's gamma level setting.

```
Fixed QTGetPixMapPtrGammaLevel (
   PixMapPtr pm
);
```

Parameters

pm

A pointer to a `PixMap` structure that has a `PixMapExtension` structure.

Return Value
On return, the gamma level previously set (or the default level) for the pixel map pointed to by the `pm` parameter.

Discussion
A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTGetPixMapPtrRequestedGammaLevel

Retrieves the current PixMap extension's gamma level setting.

```
Fixed QTGetPixMapPtrRequestedGammaLevel (
   PixMapPtr pm
);
```

Parameters

pm

A pointer to a `PixMap` structure that has a `PixMapExtension` structure.

Return Value
On return, the requested gamma level previously set (or the default level) for the pixel map pointed to by the `pm` parameter.

Discussion
A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure. The requested gamma level is used to control what gamma conversion is attempted during decompression. The requested gamma level may differ from the actual gamma level depending on the compressed data and the capabilities of the codecs involved

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTGetPixMapPtrRowBytes

Gets the rowBytes value for a pixel map accessed by a pointer.

```
long QTGetPixMapPtrRowBytes (
   PixMapPtr pm
);
```

Parameters

pm

A pointer to a `PixMap` structure.

Return Value

The `rowBytes` value.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dimmer2Effect

Dimmer2Effect.win

GreyscaleEffectSample

SoftVideoOutputComponent

VideoProcessing

Declared In

`ImageCompression.h`

QTNewGWorld

Creates an offscreen graphics world that may have a non-Macintosh pixel format.

```
OSErr QTNewGWorld (
   GWorldPtr *offscreenGWorld,
   OSType PixelFormat,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags
);
```

Parameters

offscreenGWorld

On return, a pointer to the offscreen graphics world created by this routine.

PixelFormat

The new graphics world's pixel format; see `Pixel Formats`. This function won't work with planar pixel formats; use `QTNewGWorldFromPtr` (page 137) instead. See the `ICMPixelFormatInfo` structure for a discussion of planar and chunky formats.

boundsRect

A pointer to the boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if this function creates one. If you specify 0 in the `PixelFormat` parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` structure. If you pass `NIL` in this parameter, the function uses the default color table for the pixel format that you specify in the `PixelFormat` parameter. If you set the `PixelFormat` parameter to 0, the function ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

aGDevice

A handle to a `GDevice` structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `PixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `PixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

flags

Constants (see below) that identify options available to your application. You can set a combination of these flags. If you don't wish to use any of them, pass 0 in this parameter. In this case the default behavior is to create an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, the graphics world uses an existing `GDevice` structure (if you pass 0 in the depth parameter) or creates a new `GDevice` structure, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

qteffects

qteffects.win

qtwiredspritesjr.win

VideoProcessing

Declared In
`ImageCompression.h`

QTNewGWorldFromPtr

Wraps a graphics world and pixel map structure around an existing block of memory containing an image.

```
OSErr QTNewGWorldFromPtr (
   GWorldPtr *gw,
   OSType pixelFormat,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags,
   void *baseAddr,
   long rowBytes
);
```

Parameters

gw

On entry, a pointer that isn't going to change during the lifetime of the allocated graphics world. On return, this pointer references the offscreen graphics world created by this function.

pixelFormat

The new graphics world's pixel format; see `Pixel Formats`.

boundsRect

A pointer to the boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if this function creates one. If you specify 0 in the `pixelFormat` parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` structure. If you pass `NIL` in this parameter, the function uses the default color table for the pixel format that you specify in the `pixelFormat` parameter. If you set the `pixelFormat` parameter to 0, the function ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

aGDevice

A handle to a `GDevice` structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `pixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `pixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

flags

A constant (see below) that identifies an option available to your application. If you don't wish to use this option, pass 0 in this parameter. In this case the default behavior is to create an offscreen graphics world that uses an existing `GDevice` structure (if you pass 0 in the depth parameter) or creates a new `GDevice` structure. Most constants used in creating a `GWorld` are irrelevant for this function, as its purpose is to wrap a `GWorld` around an existing block of pixels rather than to define and create a `pixmap`. See these constants:

baseAddr

The base address for the pixel data.

rowBytes

The total size of the pixel data divided by the height of the pixel map. In other words, the number of bytes in one row of pixels or the number of bytes between vertically adjacent pixels.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function wraps a `GWorld` around an existing pixel map. Note that it does not copy the `pixmap`. A subsequent call to `DisposeGWorld` will not dispose of the pixel map; it will only dispose of the `GWorld` wrapper. It is the caller's responsibility to dispose of the pixel map.

You can use this call to allocate an offscreen graphics world using special memory (such as on a video card). If you have an image in memory that belong to something else (a hardware screen buffer, a 3D card, or another file format or program), you can use this function to wrap a graphics world around the image and then use QuickTime calls on that graphics world to compress it, scale it, draw to it, and so on. If your new graphics world has a planar pixel format, you must use this call instead of `QTNewGWorld` (page 135).

Special Considerations

Do not unlock the pixels of the allocated graphics world. If your original pixels are from another graphics world then you must ensure that the source pixels are locked.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CTMClip

CTMDemo

OpenGLCompositorLab

OpenGLMovieQT

TexturePerformanceDemo

Declared In

`ImageCompression.h`

QTSetPixMapHandleGammaLevel

Sets the gamma level of a pixel map.

```
OSErr QTSetPixMapHandleGammaLevel (
   PixMapHandle pm,
   Fixed gammaLevel
);
```

Parameters

pm

A handle to a `PixMap` structure that has a `PixMapExtension` structure.

gammaLevel

The new gamma level.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the gamma level of a pixel map before compressing it so that the codec knows if it needs to do additional gamma correcting when compressing.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTSetPixMapHandleRequestedGammaLevel

Sets the requested gamma level of a pixel map.

```
OSErr QTSetPixMapHandleRequestedGammaLevel (
   PixMapHandle pm,
   Fixed requestedGammaLevel
);
```

Parameters

pm

A handle to a `PixMap` structure that has a `PixMapExtension` structure.

requestedGammaLevel

A specified gamma level or a constant (see below). See these constants:

`kQTUsePlatformDefaultGammaLevel`

`kQTUseSourceGammaLevel`

`kQTCCIR601VideoGammaLevel`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the requested gamma level of a pixel map before decompressing so that the codec knows what gamma correction is necessary when decompressing into the `PixMap` structure. The resulting gamma level can then be found by calling `QTGetPixMapHandleGammaLevel` (page 132).

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTSetPixMapHandleRowBytes

Sets the rowBytes value for a pixel map accessed by a handle.

```
OSErr QTSetPixMapHandleRowBytes (
   PixMapHandle pm,
   long rowBytes
);
```

Parameters

pm

A handle to a `PixMap` structure.

rowBytes

The `rowBytes` value to be set.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTSetPixMapPtrGammaLevel

Sets the gamma level of a pixel map.

```
OSErr QTSetPixMapPtrGammaLevel (
   PixMapPtr pm,
   Fixed gammaLevel
);
```

Parameters

pm

A pointer to a `PixMap` structure that has a `PixMapExtension` structure.

gammaLevel
The new gamma level.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the gamma level of a pixel map before compressing it so that the codec knows if it needs to do additional gamma correcting when compressing.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTSetPixMapPtrRequestedGammaLevel

Sets the requested gamma level of a pixel map.

```
OSErr QTSetPixMapPtrRequestedGammaLevel (
   PixMapPtr pm,
   Fixed requestedGammaLevel
);
```

Parameters

pm
A pointer to a `PixMap` structure that has a `PixMapExtension` structure.

requestedGammaLevel
A specified gamma level or a constant (see below). See these constants:
`kQTUsePlatformDefaultGammaLevel`
`kQTUseSourceGammaLevel`
`kQTCCIR601VideoGammaLevel`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the requested gamma level of a pixel map before decompressing so that the codec knows what gamma correction is necessary when decompressing into the `PixMap` structure. The resulting gamma level can then be found by calling `QTGetPixMapPtrGammaLevel` (page 133).

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTSetPixMapPtrRowBytes

Sets the rowBytes value for a pixel map accessed by a pointer.

```
OSErr QTSetPixMapPtrRowBytes (
   PixMapPtr pm,
   long rowBytes
);
```

Parameters

pm

A pointer to a `PixMap` structure.

rowBytes

The `rowBytes` value to be set.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTUpdateGWorld

Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world with a non-Macintosh pixel format.

```
GWorldFlags QTUpdateGWorld (
   GWorldPtr *offscreenGWorld,
   OSType PixelFormat,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags
);
```

Parameters

offscreenGWorld

On input, a pointer to an existing offscreen graphics world; upon completion, the pointer to the updated offscreen graphics world.

PixelFormat

The updated graphics world's pixel format; see `Pixel Formats`.

boundsRect

A pointer to the boundary rectangle and port rectangle for the updated offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if this function creates one. If you specify 0 in the `PixelFormat` parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. If the rectangle you specify in this parameter differs from, but has the same size as, the previous boundary rectangle, the function realigns the pixel image to the screen for optimum performance for `CopyBits`. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` structure for the updated graphics world. If you pass `NIL` in this parameter, the function uses the default color table for the pixel format that you specify in the `PixelFormat` parameter. If you set the `PixelFormat` parameter to 0, the function ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If the color table that you specify in this parameter is different from the previous color table, or if the color table associated with the `GDevice` structure that you specify in the `aGDevice` parameter is different, the function maps the pixel values in the offscreen pixel map to the new color table. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

aGDevice

A handle to a `GDevice` structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `PixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `PixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

flags

Constants (see below) that identify options available to your application. You can set a combination of these flags. If you don't wish to use any of them, pass 0 in this parameter. In this case the default behavior is to create an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, the graphics world uses an existing `GDevice` structure (if you pass 0 in the depth parameter) or creates a new `GDevice` structure, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image. See these constants:

Return Value

A constant (see below) that reports on the operation of this function.

Discussion

If the Memory Manager purged the base address for the offscreen pixel image, this function reallocates the memory but the pixel image is lost. You must reconstruct it.

Special Considerations

This function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QuadToQuadMatrix

Defines a matrix that maps between four input points and four output points.

```
OSErr QuadToQuadMatrix (
   const Fixed *source,
   const Fixed *dest,
   MatrixRecord *map
);
```

Parameters

source
A pointer to four input `FixedPoint` points.

dest
A pointer to four output `FixedPoint` points.

map
A pointer to a `MatrixRecord` structure that maps the value passed in `source` to the value passed in `dest`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export
ImproveYourImage

Declared In
`ImageCompression.h`

RectMatrix

Creates a matrix that performs the translate and scale operation described by the relationship between two rectangles.

```
void RectMatrix (
   MatrixRecord *matrix,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

matrix
: A pointer to a `MatrixRecord` structure. This function updates the contents of this matrix so that the matrix describes a transformation from points in the rectangle specified by the `srcRect` parameter to points in the rectangle specified by the `dstRect` parameter. The previous contents of the matrix are ignored.

srcRect
: A pointer to the source `Rect` structure.

dstRect
: A pointer to the destination `Rect` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview
BrideOfMungGrab
SGDataProcSample
VideoProcessing
vrmovies.win

Declared In
`ImageCompression.h`

RemoveImageDescriptionExtension

Removes a specified extension from an ImageDescription structure.

```
OSErr RemoveImageDescriptionExtension (
   ImageDescriptionHandle desc,
   long idType,
   long index
);
```

Parameters

desc
: A handle to an `ImageDescription` structure.

idType
: The type of extension to remove.

index
: The index of the extension to remove. This is a number between 1 and the count returned by `CountImageDescriptionExtensionType` (page 52).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows an application to remove a specified extension from an `ImageDescription` structure. Note that any extensions that are present in the structure after the deleted extension will have their index numbers renumbered.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ReplaceDSequenceImageDescription

Undocumented

```
OSErr ReplaceDSequenceImageDescription (
   ImageSequence seqID,
   ImageDescriptionHandle newDesc
);
```

Parameters

`seqID`
The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

`newDesc`
A handle to an `ImageDescription` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

RotateMatrix

Modifies the contents of a matrix so that it defines a rotation operation.

```
void RotateMatrix (
   MatrixRecord *m,
   Fixed degrees,
   Fixed aboutX,
   Fixed aboutY
);
```

Parameters

m

A pointer to a `MatrixRecord` structure.

degrees

The number of degrees of rotation.

aboutX

The x coordinate of the anchor point of rotation.

aboutY

The y coordinate of the anchor point of rotation.

Discussion

This function updates the contents of a matrix so that the matrix describes a rotation operation; that is, it concatenates the rotation transformations onto whatever was initially in the matrix structure. You specify the direction and amount of rotation with the `degrees` parameter. You specify the point of rotation with the `aboutX` and `aboutY` parameters.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtgraphics

qtgraphics.win

vrmovies.win

Declared In

`ImageCompression.h`

ScaleMatrix

Modifies the contents of a matrix so that it defines a scaling operation.

```
void ScaleMatrix (
   MatrixRecord *m,
   Fixed scaleX,
   Fixed scaleY,
   Fixed aboutX,
   Fixed aboutY
);
```

Parameters

m

A pointer to a `MatrixRecord` structure. The `ScaleMatrix` function updates the contents of this matrix so that the matrix describes a scaling operation; that is, it concatenates the respective transformations onto whatever was initially in the matrix structure. You specify the magnitude of the scaling operation with the `scaleX` and `scaleY` parameters. You specify the anchor point with the `aboutX` and `aboutY` parameters.

scaleX

The scaling factor applied to x coordinates.

scaleY

The scaling factor applied to y coordinates.

aboutX

The x coordinate of the anchor point.

aboutY

The y coordinate of the anchor point.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CTMClip

qtgraphics

qtgraphics.win

TexturePerformanceDemo

TextureRange

Declared In

`ImageCompression.h`

SetCompressedPixMapInfo

Stores information about a compressed image for StdPix.

```
OSErr SetCompressedPixMapInfo (
   PixMapPtr pix,
   ImageDescriptionHandle desc,
   Ptr data,
   long bufferSize,
   ICMDataProcRecordPtr dataProc,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

pix

A pointer to a `PixMap` structure that holds compressed image data.

desc

A handle to the `ImageDescription` structure that defines the compressed image.

data

A pointer to the buffer for the compressed image data. If the entire compressed image cannot be stored at this location, you may assign a data-loading function (see the `dataProc` parameter, below). This pointer must contain a 32-bit clean address.

bufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If there is no data-loading function defined for this operation, set this parameter to 0.

dataProc

A pointer to an `ICMDataProcRecord` structure. If there is not enough memory to store the compressed image, the decompressor calls an `ICMDataProc` callback that you provide, which loads more compressed data. If you do not want to assign a data-loading function, set this parameter to `NIL`.

progressProc

A pointer to an `ICMProgressProcRecord` structure. During the decompression operation, the decompressor may occasionally call an `ICMProgressProc` callback that you provide, in order to report its progress. If you do not want to assign a progress function, set this parameter to `NIL`. If you pass a value of -1, you obtain a standard progress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetCSequenceDataRateParams

Communicates information to compressors that can constrain compressed data in a particular sequence to a specific data rate.

```
OSErr SetCSequenceDataRateParams (
   ImageSequence seqID,
   DataRateParamsPtr params
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 43) or DecompressSequenceBegin (page 55).

params

A pointer to a DataRateParams structure.

Return Value

See Error Codes. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

SetCSequenceFlushProc

Assigns a data-unloading function to a sequence.

```
OSErr SetCSequenceFlushProc (
   ImageSequence seqID,
   ICMFlushProcRecordPtr flushProc,
   long bufferSize
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 43) or DecompressSequenceBegin (page 55).

flushProc

A pointer to an ICMFlushProcRecord structure. If there is not enough memory to store the compressed image, the compressor calls an ICMFlushProc callback that you provide, which unloads some of the compressed data. If you have not provided such a data-unloading function, set this parameter to NIL. In this case, the compressor writes the entire compressed image into the memory location specified by the data parameter to CompressSequenceFrame (page 46).

bufferSize

The size of the buffer to be used by the data-unloading function specified by the flushProc parameter. If you have not specified such a data-unloading function, set this parameter to 0.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion
Data-unloading functions allow compressors to work with images that cannot fit in memory. During the compression operation, the compressor calls the data-unloading function whenever it has accumulated a specified amount of compressed data. Your data-unloading function then writes the compressed data to some other device, freeing buffer space for more compressed data. The compressor starts using the data-unloading function with the next image in the sequence.

Special Considerations

There is no parameter to the `CompressSequenceBegin` (page 43) function that allows you to assign a data-unloading function to a sequence.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

SetCSequenceFrameNumber

Informs the compressor in use for the specified sequence that frames are being compressed out of order.

```
OSErr SetCSequenceFrameNumber (
   ImageSequence seqID,
   long frameNumber
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

frameNumber

The frame number of the frame that is being compressed out of sequence.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This information is necessary only for compressors that are sequence-sensitive.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

SetCSequenceKeyFrameRate

Adjusts the key frame rate for the current sequence.

```
OSErr SetCSequenceKeyFrameRate (
   ImageSequence seqID,
   long keyFrameRate
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 43) or DecompressSequenceBegin (page 55).

keyFrameRate

The maximum number of frames allowed between key frames. Set this parameter to 1 to specify all key frames, to 2 to specify every other frame as a key frame, to 3 to specify every third frame as a key frame, and so forth. The compressor determines the optimum placement for key frames based upon the amount of redundancy between adjacent images in the sequence. Consequently, the compressor may insert key frames more frequently than you have requested. However, the compressor will never place fewer key frames than is indicated by this parameter. If you set this parameter to 0, the Image Compression Manager only places key frames in the compressed sequence when you call CompressSequenceFrame (page 46), setting the `codecFlagForceKeyFrame` flag in the `flags` parameter. The compressor ignores this parameter if you have not requested temporal compression; that is, you have passed 0 for the `temporalQuality` parameter of CompressSequenceBegin (page 43).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Key frames provide points from which a temporally compressed sequence may be decompressed. Use this parameter to control the frequency at which the compressor places key frames into the compressed sequence. The new key frame rate takes effect with the next image in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetCSequencePreferredPacketSize

Sets the preferred packet size for a sequence.

```
OSErr SetCSequencePreferredPacketSize (
   ImageSequence seqID,
   long preferredPacketSizeInBytes
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

preferredPacketSizeInBytes

The preferred packet size in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

This function was added in QuickTime 2.5 to support video conferencing applications by making each transmitted packet an independently decodable chunk of data.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetCSequencePrev

Allows the application to set the pixel map and boundary rectangle used by the previous frame in temporal compression.

```
OSErr SetCSequencePrev (
   ImageSequence seqID,
   PixMapHandle prev,
   const Rect *prevRect
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

prev

A handle to the new previous image buffer. You must allocate this buffer using the same pixel depth and `ColorTable` structure as the source image buffer that you specified with the `src` parameter when you called `CompressSequenceBegin` (page 43). The compressor uses this buffer to store a previous image against which the current image is compared when performing temporal compression. The compressor manages the contents of this buffer based upon several considerations, such as the key frame rate and the degree of difference between compared images. The current image is stored in the buffer referred to by the `src` parameter to `CompressSequenceBegin`.

prevRect

A pointer to a `Rect` structure that defines the portion of the previous image to use for temporal compression. The compressor uses this portion of the previous image as the basis of comparison with the current image. This rectangle must be the same size as the source rectangle you specify with the `srcRect` parameter to `CompressSequenceBegin` (page 43). To get the boundary of a source pixel map, set this parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When you start compressing a sequence, you may assign a previous frame buffer and rectangle with the `prev` and `prevRect` parameters to `CompressSequenceBegin` (page 43). If you specified a `NIL` value for the `prev` parameter, the compressor allocates an offscreen buffer for the previous frame. In either case you may use this function to assign a new previous frame buffer.

Special Considerations

This is a very specialized function; your application should not need to call it under most circumstances.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

qteffects.win

qtwiredactions

qtwiredspritesjr.win

Declared In

`ImageCompression.h`

SetCSequenceQuality

Adjusts the spatial or temporal quality for the current sequence.

```
OSErr SetCSequenceQuality (
   ImageSequence seqID,
   CodecQ spatialQuality,
   CodecQ temporalQuality
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

spatialQuality

A constant (see below) that specifies the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A constant (see below) that specifies the desired sequence temporal quality. This parameter governs the level of compression you desire with respect to information between successive frames in the sequence. Set this parameter to 0 to prevent the compressor from applying temporal compression to the sequence.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You originally set the default spatial and temporal quality values for a sequence with `CompressSequenceBegin` (page 43). The new quality parameters take effect with the next frame in the sequence.

Special Considerations

If you change the quality settings while processing an image sequence, you affect the maximum image size that you may receive during sequence compression. Consequently, you should call `GetMaxCompressionSize` (page 104) after you change the quality settings. If the maximum size has increased, you should reallocate your image buffers to accommodate the larger image size.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceAccuracy

Adjusts the decompression accuracy for the current sequence.

```
OSErr SetDSequenceAccuracy (
   ImageSequence seqID,
   CodecQ accuracy
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

accuracy

A constant (see below) that specifies the accuracy desired in the decompressed image. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The accuracy parameter governs how precisely the decompressor decompresses the image data. Some decompressors may choose to ignore some image data to improve decompression speed. A new accuracy value takes effect with the next frame in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceDataProc

Assigns a data-loading function to a sequence.

```
OSErr SetDSequenceDataProc (
   ImageSequence seqID,
   ICMDataProcRecordPtr dataProc,
   long bufferSize
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

dataProc

A pointer to an `ICMDataProcRecord` structure. If the data stream is not all in memory when your program calls `DecompressSequenceFrame` (page 58), the decompressor calls an `ICMDataProc` callback that you provide, which loads more compressed data. If you have not provided such a data-loading function, or if you want the decompressor to stop using your data-loading function, set this parameter to `NIL`. In this case, the entire image must be in memory at the location specified by the `data` parameter to `DecompressSequenceFrame`.

bufferSize

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, set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Data-loading functions allow decompressors to work with images that cannot fit in memory. During the decompression operation the decompressor calls the data-loading function whenever it has exhausted its supply of compressed data.

Special Considerations

There is no parameter to the `DecompressSequenceBegin` (page 55) function that allows you to assign a data-loading function to a sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceFlags

Sets data loading flags.

```
OSErr SetDSequenceFlags (
   ImageSequence seqID,
   long flags,
   long flagsMask
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

flags

Flags (see below) for data loading. See these constants:

`codecDSequenceSingleField`

flagsMask

Use this field to preserve the state of any flags you do not wish to alter. If a flag (see below) is set in this field, and is not set in the `flags` parameter, it will not be changed from its current setting.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MungSaver

Declared In
`ImageCompression.h`

SetDSequenceMask

Assigns a clipping region to a sequence.

```
OSErr SetDSequenceMask (
   ImageSequence seqID,
   RgnHandle mask
);
```

Parameters

`seqID`
The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

`mask`
A handle to a clipping region in the destination coordinate system. If specified, the decompressor applies this mask to the destination image. If you want to stop masking, set this parameter to `NIL`. The new region takes effect with the next frame in the sequence.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The decompressor draws only that portion of the decompressed image that lies within the specified clipping region. You should not dispose of this region until the Image Compression Manager is finished with the sequence, or until you set the mask either to `NIL` or to a different region by calling this function again.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

SetDSequenceMatrix

Assigns a mapping matrix to a sequence.

```
OSErr SetDSequenceMatrix (
   ImageSequence seqID,
   MatrixRecordPtr matrix
);
```

Parameters

`seqID`
The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

matrix

A `MatrixRecord` structure that specifies how to transform the image during decompression. You can use this structure to translate or scale the image during decompression. To set the matrix to identity, pass `NIL` in this parameter. The new matrix takes effect with the next frame in the sequence.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The decompressor uses the matrix to create special effects with the decompressed image, such as translating or scaling the image.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceMatte

Assigns a blend matte to a sequence.

```
OSErr SetDSequenceMatte (
   ImageSequence seqID,
   PixMapHandle matte,
   const Rect *matteRect
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

matte

A handle to a `PixMap` structure 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 want to turn off the blend matte, set this parameter to `NIL`.

matteRect

A pointer to a `Rect` structure that defines the boundary rectangle for the matte. The decompressor uses only that portion of the matte that lies within the specified rectangle. This rectangle must be the same size as the source rectangle you specify with `SetDSequenceSrcRect` (page 161) or with the `srcRect` parameter to `DecompressSequenceBegin` (page 55). To use the matte's `PixMap` structure bounds as the boundary rectangle, pass `NIL` in this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The decompressor uses the matte to blend the decompressed image into the destination pixel map. The new matte and matte boundary rectangle take effect with the next frame in the sequence. You should not dispose of the matte until the Image Compression Manager has finished with the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceNonScheduledDisplayDirection

Sets the display direction for a decompress sequence.

```
OSErr SetDSequenceNonScheduledDisplayDirection (
   ImageSequence sequence,
   Fixed rate
);
```

Parameters

`sequence`

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 55) function.

`rate`

The display direction to be set. Negative values represent backward display and positive values represent forward display.

Return Value

An error code. Returns `noErr` if there is no error.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

SetDSequenceNonScheduledDisplayTime

Sets the display time for a decompression sequence.

```
OSErr SetDSequenceNonScheduledDisplayTime (
   ImageSequence sequence,
   TimeValue64 displayTime,
   TimeScale displayTimeScale,
   UInt32 flags
);
```

Parameters

sequence

Contains the unique sequence identifier that was returned by the DecompressSequenceBegin (page 55) function.

displayTime

The display time to be set.

displayTimeScale

The display time scale to be set.

flags

Not used; set to 0.

Return Value

An error code. Returns noErr if there is no error.

Availability

Available in Mac OS X v10.3 and later.

Declared In

ImageCompression.h

SetDSequenceSrcRect

Defines the portion of an image to decompress.

```
OSErr SetDSequenceSrcRect (
   ImageSequence seqID,
   const Rect *srcRect
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 43) or DecompressSequenceBegin (page 55).

srcRect

A pointer to a Rect structure that defines 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 ((**desc).width,(**desc).height), where desc refers to the ImageDescription structure you supply to DecompressSequenceBegin (page 55). If the srcRect parameter is NIL, the rectangle is set to the Rect structure in the ImageDescription.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

The decompressor acts on that portion of the compressed image that lies within this rectangle. A new source rectangle takes effect with the next frame in the sequence.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

SetDSequenceTimeCode

Sets the timecode value for a frame that is about to be decompressed.

```
OSErr SetDSequenceTimeCode (
   ImageSequence seqID,
   void *timeCodeFormat,
   void *timeCodeTime
);
```

Parameters

`seqID`
The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

`timeCodeFormat`
A pointer to a `TimeCodeDef` structure. You provide the appropriate timecode definition information for the next frame to be decompressed.

`timeCodeTime`
A pointer to a `TimeCodeRecord` structure. You provide the appropriate time value for the next frame in the current sequence.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
QuickTime's video media handler uses this function to set the timecode information for a movie. When a movie that contains timecode information starts playing, the media handler calls this function as it processes the movie's first frame. The Image Compression Manager passes the timecode information straight through to the image decompressor component. That is, the Image Compression Manager does not make a copy of any of this timecode information. As a result, you must make sure that the data referred to by the `timeCodeFormat` and `timeCodeTime` parameters is valid until the next decompression operation completes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

SetDSequenceTransferMode

Sets the mode used when drawing a decompressed image.

```
OSErr SetDSequenceTransferMode (
   ImageSequence seqID,
   short mode,
   const RGBColor *opColor
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

mode

A constant (see below) that specifies the transfer mode to be used when drawing the decompressed image. See also `Graphics Transfer Modes`. See these constants:

opColor

Contains a pointer to the color for use in `addPin`, `subPin`, `blend`, and `transparent` operations. The Image Compression Manager passes this color to QuickDraw as appropriate. If `NIL`, the opcolor is left unchanged.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

For any given sequence, the default opColor value is 50 percent gray and the default mode is `ditherCopy`. The new mode takes effect with the next frame in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetIdentityMatrix

Sets the contents of a matrix so that it performs no transformation.

```
void SetIdentityMatrix (
   MatrixRecord *matrix
);
```

Parameters

matrix

A pointer to a `MatrixRecord` structure. The function updates the contents of this matrix so that the matrix describes the identity matrix.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export
ImproveYourImage
qtspritesplus.win
vrmovies
vrmovies.win

Declared In
`ImageCompression.h`

SetImageDescriptionCTable

Updates the custom ColorTable structure for an image.

```
OSErr SetImageDescriptionCTable (
   ImageDescriptionHandle desc,
   CTabHandle ctable
);
```

Parameters

`desc`
: Contains a handle to the appropriate `ImageDescription` structure. The function updates the size of the structure to accommodate the new `ColorTable` structure and removes the old color table, if one is present.

`ctable`
: A handle to the new `ColorTable` structure. The function loads this color table into the `ImageDescription` structure referred to by the `desc` parameter. Set this parameter to `NIL` to remove a `ColorTable` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function does not change the image data, just the color table.

Special Considerations
This function is rarely used. Typically, you supply the color table when your application compresses an image, and the Image Compression Manager stores the `ColorTable` structure with the image.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites
DesktopSprites
qteffects.win
qtwiredactions
qtwiredspritesjr.win

Declared In

`ImageCompression.h`

SetSequenceProgressProc

Installs a progress procedure for a sequence.

```
OSErr SetSequenceProgressProc (
   ImageSequence seqID,
   ICMProgressProcRecord *progressProc
);
```

Parameters

`seqID`

The unique sequence identifier assigned by `CompressSequenceBegin` (page 43) or `DecompressSequenceBegin` (page 55).

`progressProc`

A pointer to an `ICMProgressProcRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows you to set an `ICMProgressProc` callback for a compression or decompression sequence, just as you can set a progress procedure when compressing or decompressing a still image.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SkewMatrix

Modifies the contents of a matrix so that it defines a skew transformation.

```
void SkewMatrix (
   MatrixRecord *m,
   Fixed skewX,
   Fixed skewY,
   Fixed aboutX,
   Fixed aboutY
);
```

Parameters

m

A pointer to the matrix for this operation. The `SkewMatrix` function updates the contents of the `MatrixRecord` structure so that it defines a skew operation; it concatenates the respective transformations onto whatever was initially in the matrix structure. You specify the magnitude and direction of the skew operation with the `skewX` and `skewY` parameters. You specify an anchor point with the `aboutX` and `aboutY` parameters.

skewX

The skew value to be applied to x coordinates.

skewY

The skew value to be applied to y coordinates.

aboutX

The x coordinate of the anchor point.

aboutY

The y coordinate of the anchor point.

Discussion

A skew operation alters the display of an element along one dimension. For example, converting a rectangle into a parallelogram is a skew operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

StdPix

Extends the grafProcs field of the CGrafPort structure to support compressed data, mattes, matrices, and pixel maps, letting you intercept image data in compressed form before it is decompressed and displayed.

```
void StdPix (
   PixMapPtr src,
   const Rect *srcRect,
   MatrixRecordPtr matrix,
   short mode,
   RgnHandle mask,
   PixMapPtr matte,
   const Rect *matteRect,
   short flags
);
```

Parameters

src

Contains a pointer to a `PixMap` structure containing the image to draw. Use `GetCompressedPixMapInfo` (page 88) to retrieve information about this structure.

srcRect

Points to a `Rect` structure that defines the portion of the image to display. This rectangle must lie within the boundary rectangle of the compressed image or within the source image. If this parameter is set to `NIL`, the entire image is displayed.

matrix

Contains a pointer to a `MatrixRecord` structure that specifies the mapping of the source rectangle to the destination. It is a fixed-point, 3-by-3 matrix.

mode

Specifies the transfer mode for the operation; see `Graphics Transfer Modes`. Note that this parameter also controls the accuracy of any decompression operation that may be required to display the image. If bit 7 (0x80) of the `mode` parameter is set to 1, the `StdPix` function sets the decompression accuracy to `codecNormalQuality`. If this bit is set to 0, the function sets the accuracy to `codecHighQuality`.

mask

Contains a handle to a clipping region in the destination coordinate system. If specified, the compressor applies this mask to the destination image. If there is no mask, this parameter is set to `NIL`.

matte

Points to a `PixMap` structure that contains a blend matte. The blend matte causes 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 there is no matte, this parameter is set to `NIL`

matteRect

Contains a pointer to a `Rect` structure that defines a portion of the blend matte to apply. This parameter is set to `NIL` if there is no matte or if the entire matte is to be used.

flags

Contains control flags (see below). See these constants:

`callOldBits`

`callStdBits`

`noDefaultOpcodes`

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

TransformFixedPoints

Transforms a set of fixed points through a specified matrix.

```
OSErr TransformFixedPoints (
   const MatrixRecord *m,
   FixedPoint *fpt,
   long count
);
```

Parameters

m

A pointer to the transformation matrix for this operation.

fpt

A pointer to the first fixed point to be transformed.

count

The number of fixed points to be transformed. These points must be stored immediately following the point specified by the `fpt` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

TransformFixedRect

Transforms the upper-left and lower-right points of a rectangle through a matrix that is specified by fixed points.

```
Boolean TransformFixedRect (
   const MatrixRecord *m,
   FixedRect *fr,
   FixedPoint *fpp
);
```

Parameters

m

A pointer to the matrix for this operation.

fr

A pointer to the `FixedRect` structure that defines the rectangle to be transformed. `TransformFixedRect` returns the updated coordinates into the structure referred to by this parameter. If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function sets the returned Boolean value to FALSE and returns the coordinates of the boundary box of the transformed rectangle. The function then updates the points specified by the `fpp` parameter to contain the coordinates of the four corners of the transformed rectangle.

fpp

A pointer to an array of four fixed points. The function returns the coordinates of the four corners of the rectangle after the transformation operation. If you do not want this information, set this parameter to `NIL`.

Return Value

If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function returns FALSE, updates the rectangle specified by the `fr` parameter to define the boundary box of the resulting rectangle, and places the coordinates of the corners of the resulting rectangle in the points specified by the `fpp` parameter. If the transformed rectangle and its boundary box are the same, the function returns TRUE.

Discussion

This function does not return any error codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

TransformPoints

Transforms a set of QuickDraw points through a specified matrix.

```
OSErr TransformPoints (
   const MatrixRecord *mp,
   Point *pt1,
   long count
);
```

Parameters

mp

A pointer to the transformation matrix for this operation.

pt1

A pointer to the first QuickDraw point to be transformed.

count

The number of QuickDraw points to be transformed. These points must be stored immediately following the point specified by the pt1 parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

TransformRect

Transforms the upper-left and lower-right points of a rectangle through a specified matrix.

```
Boolean TransformRect (
   const MatrixRecord *m,
   Rect *r,
   FixedPoint *fpp
);
```

Parameters

m

The matrix for this operation.

r

A pointer to the `Rect` structure that defines the rectangle to be transformed. The function returns the updated coordinates into the structure referred to by this parameter.

fpp

A pointer to an array of four fixed points. The `TransformRect` function returns the coordinates of the four corners of the rectangle after the transformation operation. If you do not want this information, set this parameter to `NIL`.

Return Value
If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function returns FALSE, updates the rectangle specified by the `r` parameter to define the boundary box of the resulting rectangle, and places the coordinates of the corners of the resulting rectangle in the points specified by the `fpp` parameter. If the transformed rectangle and its boundary box are the same, the function returns TRUE.

Discussion
This function does not return any error codes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

BrideOfMungGrab

BurntTextSampleCode

DrawTextCodec

ExampleCodec

Declared In
`ImageCompression.h`

TransformRgn

Applies a specified matrix to a region.

```
OSErr TransformRgn (
   MatrixRecordPtr matrix,
   RgnHandle rgn
);
```

Parameters

`matrix`
: Points to the matrix for this operation. The `TransformRgn` function currently supports only translation and scaling operations.

`rgn`
: A handle to the `MacRegion` structure to be transformed. The function transforms each point in the region according to the specified matrix

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtskins

qtskins.win

Declared In
`ImageCompression.h`

TranslateMatrix

Adds a translation value to a specified matrix.

```
void TranslateMatrix (
   MatrixRecord *m,
   Fixed deltaH,
   Fixed deltaV
);
```

Parameters

`m`
: A pointer to the `MatrixRecord` structure for this operation.

`deltaH`
: The value to be added to the x coordinate translation value.

`deltaV`

The value to be added to the y coordinate translation value.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

GroupDrawing

qtgraphics

qtgraphics.win

qtspritesplus.win

TimeCode Media Handlers

Declared In

`ImageCompression.h`

TrimImage

Adjusts a compressed image to the boundaries defined by a specified rectangle.

```
ComponentResult ADD_IMAGECODEC_BASENAME() TrimImage
```

Parameters

`desc`

A handle to the `ImageDescription` structure that describes the compressed image. On return, the compressor updates this structure to describe the resized image.

`inData`

A pointer to the compressed image data. This pointer must contain a 32-bit clean address. If the entire compressed image cannot be stored at this location, your application may provide an `ICMDataProc` callback through the `dataProc` parameter.

`inBufferSize`

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`

A pointer to an `ICMDataProcRecord` structure that references an `ICMDataProc` callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that loads more compressed data. If you have not provided such 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`

A pointer to a buffer to receive the trimmed image. The Image Compression Manager places the actual size of the resulting image into the `dataSize` field of the `ImageDescription` structure referred to by the `desc` parameter. This pointer must contain a 32-bit clean address. Your application should create a 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, in which case you use the `flushProc` parameter to identify your data-unloading function to the compressor.

outBufferSize

The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If you have not specified a data-unloading function, this parameter is ignored.

flushProc

A pointer to an `ICMFlushProcRecord` structure that references an `ICMFlushProc` callback. 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. If you have not provided such a data-unloading function, set this parameter to `NIL`. In this case, the compressor writes the entire compressed image into the memory location specified by the `data` parameter

trimRect

A pointer to a `Rect` structure that defines the desired image dimensions. On return, the function 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 left side of the image.

progressProc

A pointer to an `ICMProgressProcRecord` structure that references an `ICMProgressProc` callback. During the operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided such a progress function, set this parameter to `NIL`. If you pass a value of -1, you obtain a standard progress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

UnsignedFixMulDiv

Performs multiplications and divisions on unsigned fixed-point numbers.

```
Fixed UnsignedFixMulDiv (
   Fixed src,
   Fixed mul,
   Fixed divisor
);
```

Parameters

src

The value to be multiplied or divided.

mul

The multiplier to be applied to the value in the `src` parameter. Pass 0x00010000 if you do not want to multiply.

divisor

The divisor to be applied to the value in the `src` parameter. Pass 0x00010000 if you do not want to divide.

Return Value
The fixed-point number that is the value of the `src` parameter, multiplied by the value in the `mul` parameter and divided by the value in the `divisor` parameter. The function performs both operations before returning.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

Callbacks

Data Types

CodecComponent

Represents a type used by the Compression and Decompression API.

```
typedef Component CodecComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CodecNameSpecList

Contains a list of CodecNameSpec structures.

```
struct CodecNameSpecList {
     short            count;
     CodecNameSpec    list[1];
 };
```

Fields
`count`
Discussion
Indicates the number of compressor name structures contained in the list array that follows.

`list`
Discussion
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 calls `GetCodecNameList` (page 86).

Related Functions
`DisposeCodecNameList` (page 62)
`GetCodecNameList` (page 86)

Declared In
`ImageCompression.h`

CodecNameSpecListPtr

Represents a type used by the Compression and Decompression API.

```
typedef CodecNameSpecList * CodecNameSpecListPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ConstStrFileNameParam

Represents a type used by the Compression and Decompression API.

```
typedef ConstStr255Param ConstStrFileNameParam;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTypes.h`

DataRateParams

Communicates information to compressors that can constrain compressed data to a specific data rate.

```
struct DataRateParams {
     long      dataRate;
     long      dataOverrun;
     long      frameDuration;
     long      keyFrameRate;
     CodecQ    minSpatialQuality;
     CodecQ    minTemporalQuality;
 };
```

Fields
`dataRate`
Discussion
Specifies the bytes per second to which the data rate must be constrained.

`dataOverrun`
Discussion
Indicates the current number of bytes above or below the desired data rate. A value of 0 means that the data rate is being met exactly. If your application doesn't know the data overrun, it should set this field to 0.

`frameDuration`

Discussion
Specifies the duration of the current frame in milliseconds.

`keyFrameRate`

Discussion
Indicates the frequency of key frames. This frequency is normally identical to the key frame rate passed to the `CompressSequenceBegin` (page 43).

`minSpatialQuality`

Discussion
A constant (see below) that specifies the minimum spatial quality the compressor should use to meet the requested data rate. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

`minTemporalQuality`

Discussion
A constant (see below) that specifies the minimum temporal quality the compressor should use to meet the requested data rate.

Discussion
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.

Related Functions
`GetCSequenceDataRateParams` (page 91)
`SetCSequenceDataRateParams` (page 149)

Declared In
`ImageCompression.h`

DataRateParamsPtr

Represents a type used by the Compression and Decompression API.

```
typedef DataRateParams * DataRateParamsPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DecompressorComponent

Represents a type used by the Compression and Decompression API.

```
typedef Component DecompressorComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FixedRect

Defines the size and location of a rectangle in fixed-point numbers.

```
struct FixedRect {
     Fixed    left;
     Fixed    top;
     Fixed    right;
     Fixed    bottom;
 };
```

Fields
`left`

Discussion
The x coordinate of the upper-left corner of the rectangle.

`top`

Discussion
The y coordinate of the upper-left corner of the rectangle.

`right`

Discussion
The x coordinate of the lower-right corner of the rectangle.

`bottom`

Discussion
The y coordinate of the lower-right corner of the rectangle.

Related Functions
`TransformFixedRect` (page 168)

Declared In
`ImageCompression.h`

Fract

Represents a type used by the Compression and Decompression API.

```
typedef long Fract;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOMacOSTypes.h`

ICMPixelFormatInfo

Defines a pixel format.

```
struct ICMPixelFormatInfo {
     long            size;
     unsigned long   formatFlags;
     short           bitsPerPixel[14];
 };
```

Fields
`size`

Discussion
The size of this structure. This quantity isn't necessarily equal to `sizeof(ICMPixelFormatInfo)` because it is dependent on whether the pixel format is chunky or planar, and, if planar, the number of components (see below).

`formatFlags`

Discussion
A constant (see below) indicating the pixel format. See these constants:

`kICMPixelFormatIsPlanarMask`

`kICMPixelFormatIsIndexed`

`kICMPixelFormatIsSupportedByQD`

`bitsPerPixel`

Discussion
An array that defines the number of bits for each component. The element `bitsPerPixel[0]` contains the number of bits for the first component, `bitsPerPixel[1]` the number of bits for the second component, etc. The meaning of this parameter depends on the format flag (see below).

Discussion
You can represent a format that has from 1 to 14 discrete components using this data structure. For ARGB, there are 4 components. RGB without an alpha channel has 3 components. A component count of 15 is reserved for future expansion.

Related Functions
`ICMGetPixelFormatInfo` (page 111)
`ICMSetPixelFormatInfo` (page 115)

Declared In
`ImageCompression.h`

ICMPixelFormatInfoPtr

Represents a type used by the Compression and Decompression API.

```
typedef ICMPixelFormatInfo * ICMPixelFormatInfoPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageFieldSequence

Represents a type used by the Compression and Decompression API.

```
typedef long ImageFieldSequence;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageSequenceDataSource

Represents a type used by the Compression and Decompression API.

```
typedef long ImageSequenceDataSource;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscodeSequence

Represents a type used by the Compression and Decompression API.

```
typedef long ImageTranscodeSequence;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

OpenCPicParams

Specifies resolutions when creating images.

```
struct OpenCPicParams {
     Rect     srcRect;
     Fixed    hRes;
     Fixed    vRes;
     short    version;
     short    reserved1;
     long     reserved2;
 };
```

Fields

`srcRect`

Discussion

The optimal bounding rectangle for the resolution indicated by the `hRes` and `vRes` fields. To display a picture at a resolution other than that specified in the the `hRes` and `vRes` fields, your application should compute an appropriate destination rectangle by scaling the image's width and height by the destination resolution divided by the source resolution.

`hRes`

Discussion

The best horizontal resolution for the picture. A value of 0x0048000 specifies a horizontal resolution of 72 dpi.

`vRes`

Discussion

The best vertical resolution for the picture. A value of 0x0048000 specifies a vertical resolution of 72 dpi.

`version`

Discussion

Always set this field to -2.

`reserved1`

Discussion

Reserved; set to 0.

`reserved2`

Discussion

Reserved; set to 0.

Related Functions

`GetPictureFileHeader` (page 107)

Declared In

`ImageCompression.h`

QHdr

A Windows queue header structure.

```
struct QHdr {
     short       qFlags;
     short       pad;
     long        MutexID;
     QElemPtr    qHead;
     QElemPtr    qTail;
 };
```

Fields

`qFlags`

Discussion

Undocumented

`pad`

Discussion

Unused.

`MutexID`

Discussion

Undocumented

`qHead`

Discussion

Undocumented

`qTail`

Discussion

Undocumented

Related Functions

`CDSequenceSetSourceDataQueue` (page 33)
`Dequeue`
`Enqueue`
`InitializeQHdr`
`TerminateQHdr`

Declared In

`ImageCompression.h`

QHdrPtr

Represents a type used by the Compression and Decompression API.

```
typedef QHdr * QHdrPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Constants

StdPix Values

Constants passed to StdPix.

```
enum {
  callStdBits                    = 1,
  callOldBits                    = 2,
  noDefaultOpcodes               = 4
};
```

Declared In
`ImageCompression.h`

DSequence Flags

Constants that represent <codeVoice>DSequence</codeVoice> flags.

```
enum {
  codecDSequenceDisableOverlaySurface = (1L << 5),
  codecDSequenceSingleField     = (1L << 6),
  codecDSequenceBidirectionalPrediction = (1L << 7),
  codecDSequenceFlushInsteadOfDirtying = (1L << 8),
  codecDSequenceEnableSubPixelPositioning = (1L << 9),
  codecDSequenceDeinterlaceFields = (1L << 10)
};
```

Declared In
`ImageCompression.h`

FCompressImage Values

Constants passed to FCompressImage.

```
enum {
  codecFlagUseImageBuffer       = (1L << 0), /* decompress*/
  codecFlagUseScreenBuffer      = (1L << 1), /* decompress*/
  codecFlagUpdatePrevious       = (1L << 2), /* compress*/
  codecFlagNoScreenUpdate       = (1L << 3), /* decompress*/
  codecFlagWasCompressed        = (1L << 4), /* compress*/
  codecFlagDontOffscreen        = (1L << 5), /* decompress*/
  codecFlagUpdatePreviousComp   = (1L << 6), /* compress*/
  codecFlagForceKeyFrame        = (1L << 7), /* compress*/
  codecFlagOnlyScreenUpdate     = (1L << 8), /* decompress*/
  codecFlagLiveGrab             = (1L << 9), /* compress*/
  codecFlagDiffFrame            = (1L << 9), /* decompress*/
  codecFlagDontUseNewImageBuffer = (1L << 10), /* decompress*/
  codecFlagInterlaceUpdate      = (1L << 11), /* decompress*/
  codecFlagCatchUpDiff          = (1L << 12), /* decompress*/
  codecFlagSupportDisable       = (1L << 13), /* decompress*/
  codecFlagReenable             = (1L << 14) /* decompress*/
};
```

Constants

`codecFlagUpdatePrevious`

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

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagWasCompressed`

Indicates to your 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. This flag is set to 1 to indicate that the image was previously compressed. This flag is set to 0 if the image was not previously compressed.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagUpdatePreviousComp`

Controls whether your compressor updates the previous image buffer with the compressed image. This flag is only used with temporal compression. If this flag is set to 1, your compressor should update the previous frame buffer at the end of the frame-compression sequence, allowing your compressor to perform frame differencing against the compression results. Use the image that results from the compression operation. If this flag is set to 0, your compressor should not modify the previous frame buffer during compression.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagLiveGrab`

Indicates whether the current sequence results from grabbing live video. When working with live video, your compressor should operate as quickly as possible and disable any additional processing, such as compensation for previously compressed data. This flag is set to 1 when you are compressing from a live video source.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagDiffFrame`
: Decompress.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecFlagSupportDisable`
: Decompress.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

Declared In
`ImageCompression.h`

Color Modes

Constants that represent color modes.

```
enum {
  defaultDither                = 0,
  forceDither                  = 1,
  suppressDither               = 2,
  useColorMatching             = 4
};
enum {
  graphicsModeStraightAlpha     = 256,
  graphicsModePreWhiteAlpha     = 257,
  graphicsModePreBlackAlpha     = 258,
  graphicsModeComposition       = 259,
  graphicsModeStraightAlphaBlend = 260,
  graphicsModePreMulColorAlpha  = 261,
  graphicsModePerComponentAlpha = 272
};
enum {
  kQTTIFFUserDataPrefix         = 0x74690000, /* Added to some tag values in TIFF
 IFDs to generate user data codes.  (0x7469 is 'ti'.) */
                                        /* For example, YCbCrPositioning is tag
0x0213, so its user data code is 0x74690213. */
  kQTTIFFExifUserDataPrefix     = 0x65780000, /* Added to tag values in Exif IFDs
 to generate user data codes.  (0x6578 is 'ex'.) */
                                        /* For example, DateTimeOriginal is tag
0x9003, so its user data code is 0x65789003. */
  kQTTIFFExifGPSUserDataPrefix  = 0x67700000, /* Added to tag values in Exif GPS
IFDs to generate user data codes.  (0x6770 is 'gp'.) */
                                       /* For example, GPSAltitude is tag 0x0006,
 so its user data code is 0x6770006. */
  kQTAlphaMode                 = 'almo', /* UInt32; eg, graphicsModeStraightAlpha
 or graphicsModePreBlackAlpha */
  kQTAlphaModePreMulColor       = 'almp', /* RGBColor; used if kQTAlphaMode is
graphicsModePreMulColorAlpha */
  kUserDataIPTC                 = 'iptc'
};
```

Constants

`kQTTIFFUserDataPrefix`

Added to some tag values in TIFF IFDs to generate user data codes. (0x7469 is `'ti'`.).

Available in Mac OS X v10.1 and later.

Declared in `ImageCompression.h`.

`kQTTIFFExifUserDataPrefix`

Added to tag values in `Exif` IFDs to generate user data codes. (0x6578 is `'ex'`.).

Available in Mac OS X v10.1 and later.

Declared in `ImageCompression.h`.

`kQTTIFFExifGPSUserDataPrefix`

Added to tag values in `Exif` GPS IFDs to generate user data codes. (0x6770 is `'gp'`.).

Available in Mac OS X v10.1 and later.

Declared in `ImageCompression.h`.

`kQTAlphaMode`

UInt32; for example, `graphicsModeStraightAlpha` or `graphicsModePreBlackAlpha`.

Available in Mac OS X v10.1 and later.

Declared in `ImageCompression.h`.

`kQTAlphaModePreMulColor`

 `RGBColor`; used if `kQTAlphaMode` is `graphicsModePreMulColorAlpha`.

 Available in Mac OS X v10.1 and later.

 Declared in `ImageCompression.h`.

Declared In
`ImageCompression.h`

GetGraphicsImporterForFileWithFlags Values

Constants passed to GetGraphicsImporterForFileWithFlags.

```
enum {
  kDontUseValidateToFindGraphicsImporter = 1L << 0
};
```

Declared In
`ImageCompression.h`

QTSetPixMapPtrRequestedGammaLevel Values

Constants passed to QTSetPixMapPtrRequestedGammaLevel.

```
enum {
  kQTUsePlatformDefaultGammaLevel = 0,  /* When decompressing into this PixMap,
gamma-correct to the platform's standard gamma. */
  kQTUseSourceGammaLevel        = -1L,  /* When decompressing into this PixMap,
don't perform gamma-correction. */
  kQTCCIR601VideoGammaLevel     = 0x00023333 /* 2.2, standard television video
gamma.*/
};
```

Constants
`kQTCCIR601VideoGammaLevel`

 Gamma 2.2, for ITU-R BT.601 based video.

 Available in Mac OS X v10.0 and later.

 Declared in `ImageCompression.h`.

Declared In
`ImageCompression.h`

Document Revision History

This table describes the changes to *Compression and Decompression Reference for QuickTime*.

Date	Notes
2006-05-23	New document, based on previously published material, that describes the API for QuickTime compression and decompression operations.

Index