μTech

# JPEG

# Software Development Kit User's Guide

MuTech Corporation
85 Rangeway Road
Billerica, MA 01862
USA

For additional assistance, call MuTech's Technical Support Group at:

Telephone: 978-663-2400
Fax: 978-663-3444
Internet: support@mutech.com
Website: www.mutech.com

# Table of Contents

# 1 Introduction

## 1.1 Overview

MuTech's JPEG image compression Software Development Kit (SDK) is based on an international standard format: the Joint Photographic Experts Group (JPEG) or ISO DIS 10918-1. The SDK provides a highly optimized DCT algorithm and high speed software that compresses gray scale or color images to ratios as high as 100:1. OEMs and System integrators can quickly and easily incorporate MuTech JPEG SDK functions into their applications(for Windows or DOS). Some of these functions work independently, and some of them work in conjunction with MuTech's frame grabber products and corresponding SDKs.

The JPEG SDK includes libraries, DLLs, sample source code and build/make files. The key features of the MuTech JPEG SDK are:

- High level functions with sample code to speed up application development
- Selectable input/output file formats
- Quality factors 1<<Qf<<255
- Lossy(DCT based) or Lossless compression modes
- The code stream generated can be stored either in file or in memory buffer

MuTech JPEG SDK provides a choice of lossy or lossless compression modes.

- The Lossy(also known as Baseline) compression allows a selectable amount of loss of information during compression by adjusting the Quality Factor (QF) parameter.
- The lossless mode compresses the image without introducing any distortion. The decompressed image will be exactly the same as the original.

## 1.2 Baseline Compression and Decompression

The Baseline compression/decompression is based on Discrete Cosine Transform (DCT), Quantization, and Huffman coding. Block diagrams of these processes are shown in Figure 1-1 and Figure 1-2.

### Compression

To achieve the best compression results, JPEG handles colors as separate components such as RGB and YCbCr (YUV). For Baseline, MuTech JPEG always converts RGB images into YCbCr color space for compression. The decompressed image can be converted back from YCbCr to RGB, if so desired, by the application.

After the pre-process of color space conversion (if necessary), the image will go through the following procedures (as shown in Figure 1-1) for compression:

1 An image is divided up into 8x8 blocks. Each block is transformed into the frequency domain using Forward DCT. The results of the transformation is a set of 64 DCT coefficients. One of these values at location (0,0) is referred to as the DC coefficient and the rest as the AC coefficients.

2 Each of these DCT coefficients is quantized using one of the corresponding values from the quantization table. This quantization table can be scaled to generate different amount of compression. The scaling process is controlled by the Quality Factor provided by the user.

3 Huffman code tables are then used to encode the entropy of the quantized DCT coefficients. These Huffman code tables are pre-defined inside the MuTech JPEG SDK.

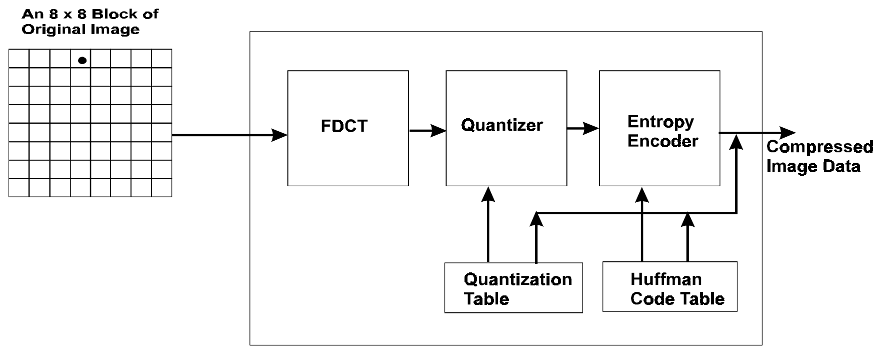

**Figure 1-1: Baseline Encoding Diagram**

### Decompression

Reversing the compression process, an image is restored from the compressed code stream as follows (Figure 1-2):

1 The compressed code stream is decoded into DCT coefficients by using the same Huffman code tables used in the compression process.

2 The DCT coefficients are dequantized using the same quantization tables as in the compression process.

3 The group of 64 dequantized DCT coefficients are transformed back into 8x8 blocks of pixels using Inverse DCT.

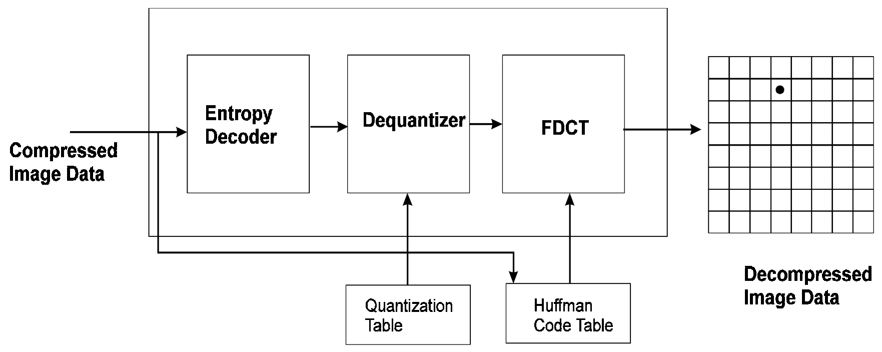

**Figure 1-2: Baseline Decoding Diagram**

After the decompression, the result will be converted into an RGB color image if the user desires. The restored image will not be exactly the same as the original. The degree of distortion<N>is inversely proportional to the Quality Factor spesified during compression. The higher the Quality Factor, the less the image distortion. Please refer to Appendix B for details on Baseline compression and decompression.

## 1.3 Lossless Compression and Decompression

Lossless Codec (Compression and Decompression) is less complicated than Baseline Codec. The image is not divided into blocks, and there is no DCT transformation, quantization, or color space conversion. JPEG uses the original image’s color components for compression and decompression.

The compression is based on neighborhood prediction and Huffman coding. Block diagrams for lossless compression and decompression are shown in Figure 1-4 and Figure 1-5 respectively.

### Compression

Lossless Codec uses a predictive coding technique based on the “Differential Pulse Code Modulation” (DPCM) model. The prediction error from the DPCM is encoded by the same Huffman coding used by Baseline codec. The following are the steps for Lossless compression (Figure 1-4.) :

1 A predictor combines the reconstructed values of up to three neighborhood samples at positions a, b and c to form a prediction of the sample at position x as shown in Figure 1-3.

2 This prediction is subtracted from the actual value of the sample at the position x generating the difference.

3 The difference is then coded by Huffman coding.

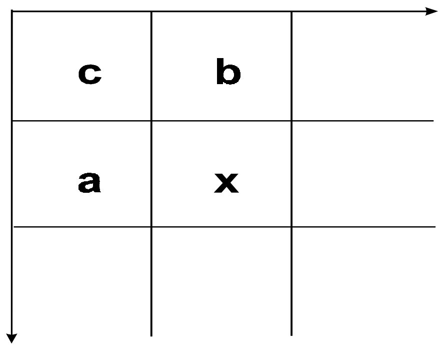

**Figure 1-3: Three-sample Prediction Neighborhood**

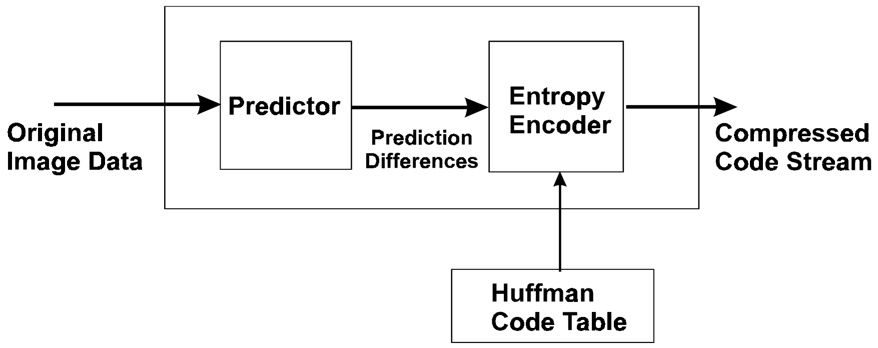

**Figure 1-4: Lossless Encoding Diagram**

## Decompression

The procedure below restores an image from a compressed code stream(Figure 1-5):

1 The Huffman code is decoded for the prediction difference of each pixel.

2 The prediction value for each pixel is calculated. This is based on the same predictor used in the encoding process and the same reconstructed neighborhood samples, for example a, b and c as shown in Figure 1-3.

3 To restore the original image, the prediction value of each pixel (generated from Step 2) is added to the prediction difference of each pixel (decoded from Step 1).

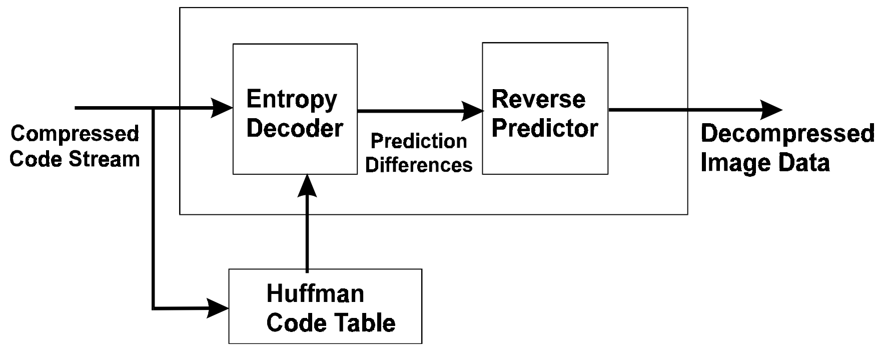

**Figure 1-5: Lossless Decoding Diagram**

After the decompression, the restored image will be exactly the same as the original. Refer to Appendix C for definitions of different predictors used by JPEG lossless compression.

# 2 Using MuTech JPEG SDK

The MuTech JPEG SDK is provided separately for different platforms, such as DOS, WIN16 and WIN32. SDK's for other platforms may also be available. Call MuTech Corporation for information.

The MuTech JPEG SDK adds JPEG compression/decompression capabilities to other MuTech products. It currently supports the IV-4XX product line together with the IV-4XX SDK. Support for other MuTech products will become available in the future. Generally, the MuTech JPEG SDK is a software package which can be used either alone or with any other MuTech SDKs.

The following sections describe the contents and the organizations of the JPEG SDK for different platforms.

## 2.1 DOS 16 Bit Library

The DOS version of the MuTech JPEG SDK provides a group of 16 bit libraries, MTJPEG.LIB, FIOLIB.LIB, and IV4JPEG.LIB, which are based on Microsoft Visual C/C++ 1.5x and work only with the MSC compiler. Only large memory mode libraries are provided.

Once installed, the following files can be found in the sub-directories:

```
X:\MTJPEG\INC\
           MTJPEG.H
           PORTABLE.H
           STD_DEF.H

X:\MTJPEG\LIB\
           MTJPEG.LIB
           FIOLIB.LIB
           IV4JPEG.LIB

X:\MTJPEG\DOC\README.DOS

X:\MTJPEG\SAMPLES\JPEG\DOS\
           DF2FCMP.C
           DF2FCMP.MAK
           DF2FDEC.C
           DF2FDEC.MAK
```

The following files are also included in the MuTech JPEG SDK, but are expected to be used with the IV-4XX SDK for DOS:

```
X:\MTJPEG\SAMPLES\IV4JPEG\DOS\
           DJPEG.C
           DJPEG.MAK
           COURF.FON
```

Please refer to README.DOS for information on how to build and run the samples.

## 2.2 Windows 16 Bit DLL

The WIN16 version of the MuTech JPEG SDK provides a 16 bit Dynamic Link Library (DLL), MTJPEG16.DLL, and an import library, MTJPEG16.LIB, which are based on Microsoft Visual C/C++ 1.5x and which work under both Windows 3.1x and Windows 9X. Since MTJPEG16.DLL is a standard Windows DLL, is should work with other compilers as well.

Once installed, the following files can be found in the sub-directories:

```
X:\MTJPEG\INC\
           MTJPEG.H
           PORTABLE.H
           STD_DEF.H

X:\MTJPEG\LIB\
           MTJPEG16.LIB
           MTJPEG16.DLL

X:\MTJPEG\DOC\README.W16

X:\MTJPEG\SAMPLES\JPEG\WIN16\
           WMEMORY.C
           WMEMORY.H
           WMEMORY.RC
           WMEMORY.DEF
           WMEMORY.MAK
```

The following files are also included in the MuTech JPEG SDK, but are expected to be used with the IV-4XX SDK for Windows 3.1x:

```
X:\MTJPEG\SAMPLES\IV4JPEG\WIN31\
           WJPEG.C
           WJPEG.H
           WJPEG.RC
           WJPEG.DEF
           WJPEG.MAK
```

Please refer to README.W16 for information on how to build and run the samples.

## 2.3 Windows 32 Bit DLL

The WIN32 version of the MuTech JPEG SDK provides a 32 bit Dynamic Link Library (DLL), MTJPEG32.DLL, and an import library, MTJPEG32.LIB, which are based on Microsoft Visual C/C++ 2.x or later and chich work under both Windows 9X and Windows NT. Since MTJPEG32.DLL is a standard 32 bit Windows DLL, it should work with other compilers as well.

Once installed, the following files can be found in the sub-directories:

```
X:\MTJPEG\INC\
           MTJPEG.H
           PORTABLE.H
           STD_DEF.H

X:\MTJPEG\LIB\
           MTJPEG32.LIB
           MTJPEG32.DLL

X:\MTJPEG\DOC\README.W32

X:\MTJPEG\SAMPLES\JPEG\WIN32\
           CF2FCMP.C
           CF2FCMP.MAK
           CF2FCMP.DSW
           CF2FCMP.DSP
           CF2FDEC.C
           CF2FDEC.MAK
           CF2FDEC.DSW
           CF2FDEC.DSP

X:\MTJPEG\SAMPLES\JPEG\WIN32\
           CMEMORY.C
           CMEMORY.H
           CMEMORY.RC
           CMEMORY.DEF
           CMEMORY.MAK
           CMEMORY.DSW
           CMEMORY.DSP
```

The following files are also included in the MuTech JPEG SDK, but are expected to be used with the IV-4XX SDK for Windows 9X or for Windows NT:

```
X:\MTJPEG\SAMPLES\IV4JPEG\WIN95\
           CJPEG.C
           CJPEG.H
           CJPEG.RC
           CJPEG.DEF
           CJPEG.MAK
           CJPEG.DSW
           CJPEG.DSP

X:\MTJPEG\SAMPLES\IV4JPEG\WINNT\
           NJPEG.C
           NJPEG.H
           NJPEG.RC
           NJPEG.DEF
           NJPEG.MAK
           NJPEG.DSW
           NJPEG.DSP
```

Please refer to README.W32 for information on how to build and run the samples.

# 3 Function References

The APIs documented in this chapter have identical interfaces for all platforms supported.

The functions in MuTech JPEG SDK have been divided into several groups, the Application Level Group, the MuTech Product Support Group and the Miscellanous Utility Group. Each will be discussed in this chapter.

Application Level Group provides easy-to-use interfaces for most of the applications to compress/decompress images between files and the system memory. This group of functions can be used in any applications that may or may not relate to MuTech products.

MuTech Product Support Group includes functions which must be used with other MuTech products. Generally, these routines will perform compression or decompression between Video Buffer and disk files. Currently, only IV-4XX product line are supported. Supports for other product line may be available in the future.

Miscellaneous Utility Group provides miscellaneous supports to applications, such as RGB and YCbCr color space conversion or file information inquire functions.

There is a lower level function group called the Developer Level Group in the JPEG SDK. This group is the base for all higher level function groups, e.g., the Application Level and the MuTech Product Support groups. To use the routines in the Developer Level Group requires using complicated call back schemes. This group of functions are not domcumented here.

## 3.1 Application Level Group

This function group uses the following naming conventions.

Compression

JPEG_**Source**2**Destination**_Compress

- **Source -** can be File or Memory in which the input image data are located. The data format can be 8 bit Black/White, 16 bit YCrCb (YUYV), or 24 bit RGB (in some cases).
- **Destination -** can be File or Memory in which the output code stream will be stored in JPEG format.

Decompression

JPEG_**Source**2**Destination**_Decompress

- **Source -** can be File or Memory in which the input code stream is located in JPEG format.
- **Destination -** can be File or Memory in which the output iamge data will be stored. The data format can be 8 bit Black/White, 16 bit YCbCr (YUYV), or 24 bit RGB (in some cases).

Parameter used to return data to caller

To distingush any parameter which can be used to return data/information to the caller, we put a small left arrow in front of the parameter. This format is shown below.

⇦ **OutputParameter**

## 3.1.1 JPEG_InquireSDKVersion

SYNOPSIS

```
#include “mtjpeg.h”
UWORD MTPROCALL JPEG_InquireSDKVersion(void);
```

DESCRIPTION

Returns the current JPEG SDK version.

PARAMETERS

None

RETURN VALUE

A 16 bit hex value, the higher 8 bits represent the major version number and the lower 8 bits represent the minor version number.

SIDE EFFECT

None.

EXAMPLE

```
UWORD SDKVersion;

// inquires SDK version
SDKVersion = JPEG_InquireSDKVersion();
// reports SDK version
printf(“SDK Version is %x.%x, SDKVersion >> 8,
       SDKVersion & 0x00ff);
```

## 3.1.2 JPEG_InquireCodeStreamSize

SYNOPSIS

```
#include “mtjpeg.h”
ULONG MTPROCALL JPEG_InquireCodeStreamSize();
```

DESCRIPTION

Returns the total number of bytes of the code stream processed by the last compression/decompression function call.

PARAMETERS

None

RETURN VALUE

A 32 bit integer vaule which is the number of bytes of the code stream processed by the last compression/decompression function call.

COMMENTS

This function can be called following any one of the compression/ decompression function calls. It will return the code stream size processed by the last compression/decompression function. Or, if none of the compression/decompress functions has been called, this function should return a value 0. The code stream size can be used to calculate the compression ratio.

SIDE EFFECT

None.

EXAMPLE

```
ULONG CodeSize;

// called a compression function
JPEG_File2File_Compress(...);
// get code stream size
CodeSize = JPEG_InquireCodeStreamSize();
// now, CodeSize contains the number of bytes
// of the compressed file.

// another example

// called a decompression function
JPEG_File2File_Decompress(...);
// get code stream size
CodeSize = JPEG_InquireCodeStreamSize();
// now, CodeSize contains the number of bytes
// from the input file that has been decoded.
```

## 3.1.3 JPEG_File2File_Compress

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL JPEG_File2File_Compress(
            char FAR *InFile,
            CompressParameter cmpPara,
            char FAR *OutFile);
```

DESCRIPTION

Compresses an image file in BMP, TGA, or TIF format into a JPG file.

PARAMETERS

**InFile**

A pointer to a string that specifies the input file name which may contain a path and must have an extension of BMP, TGA, or TIF.

**cmpPara**

Specifies the compression parameters in the **CompressParameter** structure defined below:

```
typedef struct COMPRESSPARAMETER {
       BYTE CompressMethod;
       BYTE QualityFactor;
       BYTE CompressedDataFormat;
       BYTE FAR *Comments;
       UWORD CommentLength;
} CompressParamter;
```

The member of the structure are explained below:

- `CompressMethod`

  Specifies the compression method which can be one of the following pre-defined macros:

  *`JPEG_LOSSY`*
  Uses the JPEG Baseline Compression.

  *`JPEG_LOSSLESS`*
  Uses the JPEG Lossless Compression.

- `QualityFactor`

  Specifies a Quality Factor, which ranges from 1 to 255, when JPEG_LOSSY is used.

  When JPEG_LOSSLESS is used. it selects one of the predictors, to be used in compression. The valid value is from 1 to 7.

- `CompressedDataFormat`

  This entry is reseved and must be filled with the following pre-defined macro:

  *`JPEG_AS_SOURCE`*
  See comments for details.

- `Comments`

  Pointer to a byte buffer that contains application related information, such as comment, time stamp, etc. The contents in the buffer will be copied to the compressed file using a JPEG Comment Marker. The user can retrieve this information by calling either the **MTInquireJPGVals( )** or one of the decompression functions that decompresses from a JPG file.

- `CommentLength`

  Specifies the length of the information in bytes. If the contents is longer than **CommentLength**, the extra part will be dropped. The user must ensure that the **CommentLength** is not longer than the size of the **Comments** buffer.

**OutFile**

A pointer to a string that specifies the output file name which may contain a path and must have an extension of JPG.

RETURN VALUE

OK

The function completed successfully.

PARA_ERROR

One of the parameter is invalid.

COMMENTS

For lossy (baseline) compression, the Quality Factor is used to control the compressed image quality. The higher the Quality Factor, the better the compressed image quality and the lower the compression ratio, and vice versa. To adjust the compression ratio and compressed image quality, MuTech recommends using 128 as the start value. Usually this value will generate acceptable results for both compression ratio and compressed image quality. The user may experiment with different settings and choose the best one.

For lossless compression, a Predictor Number is used. For details about the definition of the Predictor Number, refer to Appendix C.

This function supports three type of input files, which are BMP, TGA and TIF. Moreover, only 8 bit Black/White and 24 bit RGB image can be compressed except for TIFF file, in which case, the YCbCr422 format is also supported. The output file must be JPG file.

Compression Data Format will be determined internally by the source data file and the Compress Method. If the input data is 8 bit Black/White, the Compressed Data Format will be 8 bit Black/White. If the input Data is 24 bit RGB, for lossless compression the Compressed Data format will be 24 bit RGB, for baseline compression the input data will be converted to YCbCr422, and the Compressed Data Format will be YCbCr422. For YCbCr422 input data format, only lossy compression is supported and it will be compressed in the YCbCr422 format.

SIDE EFFECT

None.

SEE ALSO

`JPEG_File2File_Decompress( ), MTInquireJPGVals( )`

EXAMPLE

```
CompressParameter cPara;

// fill out the input parameter structure
cPara.CompressMethod = JPEG_LOSSY;
cPara.QulityFactor = 128;
cPara.CompressedDataFormat = JPEG_AS_SOURCE;
cPara.Comments = NULL;
cPara.CommentLength = 0;

// compress
if (JPEG_File2File_Compress(“test.bmp”, cPara,
                         “test.jpg”) != OK)
{
   // error occurs
}
.
.
// decompress
if (JPEG_File2File_Decompress(“test.jpg”,
                      DATA_AS_COMPRESSED,
                      “test.tga”,
                      NULL, NULL) != OK)
{
   // error occurs
}
```

## 3.1.4 JPEG_File2File_Decompress

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL JPEG_File2File_Decompress(
            char FAR *InFile,
            BYTE PixelFormat,
            char FAR *OutFile,
            BYTE FAR *Comments,
            UWORD FAR *cLength);
```

DESCRIPTION

Decompresses an image file in JPG format into an image file of BMP, TGA or TIF format.

PARAMETERS

**InFile**

A pointer to a string that specifies the input file name which may contain a path and must have an extension of JPG.

**PixelFormat**

This parameter must be filled with one of the pre-defined macros:

*DATA_AS_COMPRESSED*
Used for most cases, see comments for details.

*DATA_16bit_YCbCr422*

This is an exception that may be used only when the compression was lossy and the OutFile has an extension of TIF. See comments for details.

**OutFile**
A pointer to a string that specifies the output file name which may contain a path and must have an extension of BMP, TGA or TIF.

⇦ **Comments**
Pointer to a byte buffer that will be used to hold the JPEG comment information if one exists in the compressed code stream. A NULL pointer could be used if the user has no interest of the comment content at the time of calling. If a buffer pointer is specified, its length must be specified by **cLength**, see below for detailed description. It is the caller's responsibility to allocate enough space for the buffer, otherwise the extra part of the **Comments** field will be discarded. The user can also retrieve the comment information by **MTInquireJPGVals**( ) function.

⇦ **cLength**
Pointer to an UWORD variable that can be used in a number of ways described below:

- points to NULL - the user is not interested in any information in the comment field. The function will not look for Comment Marker and totally ignore the **Comments** buffer.
- points to a variable with the value 0 - The user is interested in only the length of the comment field. If there is no Comment Marker found, the return value will be 0 (unchanged). Otherwise, this function will return the actual length of the comment field in the variable pointed to. The **Comments** buffer will be ignored.
- points to a variable with non-zero value - The user is interested in both the length and the content of the comment field. In this case, the **Comments** buffer must be allocated and supplied, and the value of the variable must be equal to the size of the **Comments** buffer. If there is no Comment Marker found, the return value will be 0. Otherwise, this function will copy the comment field into the buffer pointed by **Comments**, and return the actual length of the comment field in the variable pointed to.

RETURN VALUE

OK
The function completed successfully.

PARA_ERROR
One of the parameter is invalid.

COMMENTS

In most cases, the DATA_AS_COMPRESSED is used for **PixelFormat**. The Decompressed Data Format will be determined by how the data was compressed in the **InFile**. If the Compressed Data Format is 8 bit Black/White, the Decompressed Data Format will be 8 bit Black/White. If the Compressed Data Format is 24 bit RGB, the Decompressed Data Format will be 24 bit RGB (for lossless). Or, for the Compressed Data Format of YCbCr422, the Decompressed Data Format will be converted to RGB (for lossy), except if the **OutFile** is TIF and if the compression was lossy, then the output format can be the YCbCr422 format, which must be specified by using the pre-defined macro DATA_16bit_YCbCr.

In General, there two ways that the application can retrieve the comment field from a compressed code stream. The first is to call **MTInquireJPGVals**( ) function to find out the information about the comment field, then allocate the buffer of the right size and retrieve the comment field during the decompression function call. The second way is to blindly allocate a buffer which is large enough to hold the maximum anticipated size of the comment field, then calls this function to get the information needed.

SIDE EFFECT

None.

SEE ALSO

`JPEG_File2File_Compress( ), MTInquireJPGVals( )`

EXAMPLE

See sample for **JPEG_File2File_Compress**( ).

## 3.1.5 JPEG_File2Memory_Compress

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL JPEG_File2Memory_Compress(
            char FAR *InFile,
            CompressParameter cmpPara,
            BYTE HUGE *jpegBuffer,
            ULONG FAR *bufferLength);
```

DESCRIPTION

Compresses an image file in BMP, TGA, or TIF format into a memory buffer in JPG file format.

PARAMETERS

**InFile**

A pointer to a string that specifies the input file name which may contain a path and must have an extension of BMP, TGA, or TIF.

**cmpPara**

Specifies the compression parameters in the **CompressParameter** structure defined below:

```
typedef struct COMPRESSPARAMETER {
       BYTE CompressMethod;
       BYTE QualityFactor;
       BYTE CompressedDataFormat;
       BYTE FAR *Comments;
       UWORD CommentLength;
   } CompressParamter;
```

The member of the structure are explained below:

- `CompressMethod`

  Specifies the compression method which can be one of the following pre-defined macros:

  *`JPEG_LOSSY`*
  Uses the JPEG Baseline Compression.

  *`JPEG_LOSSLESS`*
  Uses the JPEG Lossless Compression.

- `QualityFactor`

  Specifies a Quality Factor, which ranges from 1 to 255, when JPEG_LOSSY is used.

  When JPEG_LOSSLESS is used. it selects one of the predictors, to be used in compression. The valid value is from 1 to 7.

- `CompressedDataFormat`

  This entry is reseved and must be filled with the pre- defined macro:

  *`JPEG_AS_SOURCE`*
  See comments for details.

- `Comments`

  Pointer to a byte buffer that contains application related information, such as comment, time stamp, etc. The contents in the buffer will be copied to the compressed **jpegBuffer** using a JPEG Comment Marker. The user can retrieve this information by calling either the **MTInquireJPGInMemoryVals**( ) or one of the decompression functions that decompresses from a JPG memory buffer.

- `CommentLength`

  Specifies the length of the information in bytes. If the contents is longer than **CommentLength**, the extra part will be dropped. The user must ensure that the **CommentLength** is not longer than the size of the **Comments** buffer.

⇦ **jpegBuffer**
A pointer to a buffer of bytes which will be used to store the compressed code stream in JPG format.

**bufferLength**
A ULONG variable that specifies the length of the buffer. The requirement of the buffer length depends on the compression ratio which is, in turn, depending on the Quality Factor parameter. It is the caller’s responsibility to allocate enough space to accommodating the code stream. A rule of thumb is to allocate a buffer the same size of the original image.

RETURN VALUE

OK
The function completed successfully.

PARA_ERROR
One of the parameters is invalid.

COMMENTS

This is functionally identical to **JPEG_File2File_Compress**( ). The only difference is that this function stores the compressed code stream into a memory buffer instead of a disk file. The codes generated by these two functions are identical. Please refer to **JPEG_File2File_Compress**( ) for more comments.

SIDE EFFECT

None.

SEE ALSO

```
JPEG_File2File_Compress( )
JPEG_Memory2File_Decompress( )
MTInquireJPGInMemoryVals( )
```

EXAMPLE

```
CompressParameter cPara;
BYTE HUGE *buffer;
UWORD dx, dy, bp;
ULONG length;

// fill out the input parameter structure
cPara.CompressMethod = JPEG_LOSSY;
cPara.QulityFactor = 128;
cPara.CompressedDataFormat = JPEG_AS_SOURCE;
cPara.Comments = NULL;
cPara.CommentLength = 0;

// assume the compression ratio is greater than 1:1
// the following buffer length is considered most
// conservative that assume nothing is compressed
MTInquireBMPVals(“test.bmp”, &dx, &dy, &bp);
bp = bp / 8;   // bytes per pixel
length = (ULONG)dx * (ULONG)dy * (ULONG)bp;

// allocate buffer
buffer = (BYTE HUGE *)MTMALLOC(length);
// compress
if (JPEG_File2Memory_Compress(“test.bmp”,
                      cPara,
                      buffer,
                      &length) != OK)
{
       // error occurs
}
.
.
// Now, the length contains the actual size of
// the code stream in bytes.
// decompress
if (JPEG_Memory2File_Decompress(buffer,
                       length,
                       DATA_AS_COMPRESSED,
                       “test.tga”,
                       NULL, NULL) != OK)
```

```
{
       // error occurs
}

// release buffer
MTFREE(buffer);
```

### 3.1.6 JPEG_Memory2File_Decompress

SYNOPSIS

```
#include "mtjpeg.h"
int MTPROCALL JPEG_Memory2File_Decompress(
                  BYTE HUGE *jpegBuffer,
                  ULONG bufferLength,
                  BYTE PixelFormat,
                  char FAR *OutFile,
                  BYTE FAR *Comments,
                  UWORD FAR *cLength);
```

DESCRIPTION

Decompresses a JPG format memory buffer into a BMP, TGA or TIF file.

PARAMETERS

**jpegBuffer**

A pointer to a BYTE buffer that holds the compressed code stream in JPG format.

**bufferLength**

Specifies the length of the buffer. The **bufferLength** defines the byte number of code in the buffer. The **bufferLength** usually has a value returned from previous call to **JPEG_File2Memory_Compress**( ) function.

**PixelFormat**

This parameter must be filled with one of the pre-defined macros:

*DATA_AS_COMPRESSED*
Used for most cases, see comments for details.

*DATA_16bit_YCbCr422*
This is an exception that may be used only when the compression was lossy and the **OutFile** has an extension of TIF. See comments for details.

**OutFile**

A pointer to a string that specifies the output file name which may contain a path and must have an extension of BMP, TGA or TIF.

⇦ **Comments**

Pointer to a byte buffer that will be used to hold the JPEG comment information if one exists in the compressed code stream. A NULL pointer could be used if the user has no interest of the comment content at the time of calling. If a buffer pointer is specified, its length must be specified by **cLength**, see below for detailed description. It is the caller's responsibility to allocate enough space for the buffer, otherwise the extra part of the **Comments** field will be discarded. The user can also retrieve the comment information by **MTInquireJPGInMemoryVals**( ) function.

⇦ **cLength**

Pointer to an UWORD variable that can be used in a number of ways described below:

- points to NULL **-** the user is not interested in any information in the comment field. The function will not look for Comment Marker and totally ignore the **Comments** buffer.
- points to a variable with the value 0 **-** The user is interested in only the length of the comment field. If there is no Comment Marker found, the return value will be 0 (unchanged). Otherwise, this function will return the actual length of the comment field in the variable pointed to. The **Comments** buffer will be ignored.
- points to a variable with non-zero value **-** The user is interested in both the length and the content of the comment field. In this case, the **Comments** buffer must be allocated and supplied, and the value of the variable must be equal to the size of the **Comments** buffer. If there is no Comment Marker found, the return value will be 0. Otherwise, this function will copy the comment field into the buffer pointed by **Comments**, and return the actual length of the comment field in the variable pointed to.

RETURN VALUE

OK

The function completes successfully.

PARA_ERROR

One of the parameter is invalid.

COMMENTS

This function is functionally identical to **JPEG_File2File_Decompress**( ). The only difference is that this function reads the compressed code stream from a memory buffer instead of from a disk file. The decompressed image files generated by these two functions are identical, if they have been compressed by the same controlling parameters. Please refer to **JPEG_File2File_Deompress**( ) for detailed comments.

In General, there two ways that the application can retrieve the comment field from a compressed code stream. The first is to call **MTInquireJPGInMemoryVals**( ) function to find out the information about the comment field, then allocate the buffer of the right size and retrieve the comment field during the decompression function call. The second way is to blindly allocate a buffer which is large enough to hold the maximum anticipated size of the comment field, then calls this function to get the information needed.

SIDE EFFECT

None.

SEE ALSO

`JPEG_File2Memory_Compress( ),`

```
JPEG_File2File_Decompress( ),
MTInquireJPGInMemoryVals( )
```

EXAMPLE

See sample for **JPEG_File2Memory_Compress**( ).

## 3.1.7 JPEG_Memory2File_Compress

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL JPEG_Memory2File_Compress(
                  DataFrame dataFrame,
                  ROI region,
                  CompressParameter cmpPara,
                  BYTE FAR *OutFile);
```

DESCRIPTION

Compresses an image from a pre-defined Data Frame in system memory into a JPG file.

PARAMETERS

**dataFrame**

Specifies an image in system memory as below:

```
typedef struct DATAFRAME {
       BYTE HUGE *buffer;
       UWORD dx;
       UWORD dy;
       BYTE UpDownFlip;
       BYTE PixelFormat;
       BYTE Reserved;
    } DataFrame;
```

The members of the structure are explained below:

- `buffer`

  Point to a byte buffer that contains the image data. The image size and format are specified by the other members of this structure. The buffer length must be calculated as:

  ```
  if (PixelFormat == DATA_8bit_BW) bp = 1;
  else if (PixelFormat == DATA_16bit_YCbCr422 ||
         PixelFormat == DATA_16bit_YONLY)
         bp = 2;
  length = dx * dy * bp;
  ```

- `dx`

  Specifies the number of pixels per line in the Data Frame.

- `dy`

  Specifies the number of lines in the Data Frame.

- `UpDownFlip`

This flag specifies the direction of compressing, inverse vertically if set to 1.

- `PixelFormat`

  Specifies the pixel format for the Data Frame. Only the following formats are supported:

  *`DATA_8bit_BW`*
  8 bit per pixel Black/White format.

  *`DATA_16bit_YCbCr422`*
  16 bit per pixel YCbCr422 format.

  *`DATA_16bit_YONLY`*
  16 bit per pixel YCbCr422 format. But only the Y component is relavent in this case, the Cb and Cr component are treated as dummies.

**region**

Specifies a Region Of Interest by the **ROI** structure:

```
typedef struct ROI {
       UWORD sx;
       UWORD sy;
       UWORD dx;
       UWORD dy;
    } ROI;
```

The members of the structure are explained below:

- `sx, sy`

  Specify, in term of pixels, the offset of the upper left corner of the region in a Data Frame.

- `dx, dy`

  Specify, in term of pixels and lines, the width and height of the region in a Data Frame.

Note, a region must be assoicated with a Data Frame in order to be completely defined. Only the image in the **region** will be compressed. To compress the entire Data Frame, define the region equal to the frame.

**cmpPara**

Specifies the compression parameters in the **CompressParameter** structure defined below:

```
typedef struct COMPRESSPARAMETER {
       BYTE CompressMethod;
       BYTE QualityFactor;
       BYTE CompressedDataFormat;
       BYTE FAR *Comments;
       UWORD CommentLength;
       } CompressParamter;
```

The member of the structure are explained below:

- `CompressMethod`

  Specifies the compression method which can be one of the following pre-defined macros:

  *`JPEG_LOSSY`*
  Uses the JPEG Baseline Compression.

  *`JPEG_LOSSLESS`*
  Uses the JPEG Lossless Compression.

- `QualityFactor`

  Specifies a Quality Factor, which ranges from 1 to 255, when JPEG_LOSSY is used.

  When JPEG_LOSSLESS is used. it selects one of the predictors, to be used in compression. The valid value is from 1 to 7. For details about predictor, refer to Appendix C.

- `CompressedDataFormat`

  Specifies the Compressed Data Format with one of the following pre-defined macros:

  *`JPEG_8bit_BW`*
  Specifies that the Compressed Data Format will be 8 bit per pixel Black/White format regardless what the Pixel Format is selected in the Data Frame. If the Pixel Format in the Data Frame is DATA_16bit_YCbCr or DATA_16bit_YONLY, the image data will be converted to DATA_8bit_BW before compression.

  *`JPEG_16bit_YCbCr422`*
  Specifies that the Compressed Data Format will be 16 bit per pixel YCbCr422 format. This parameter is valid only when the input data format is DATA_16bit_YCbCr422.

  *`JPEG_AS_SOURCE`*
  The Compressed Data Format will be determined by input data, i.e., Pixel Format in the Data Frame. When the Pixel Format is DATA_8bit_BW, the Compressed Data Format will be JPEG_8bit_BW. When the Pixel Format is DATA_16bit_YCbCr422 or DATA_16bit_YONLY, the Compressed Data Format will be JPEG_16bit_YCbCr422.

- `Comments`

  Pointer to a byte buffer that contains application related information, such as comment, time stamp, etc. The contents in the buffer will be copied to the compressed file using a JPEG Comment Marker. The user can retrieve this information by calling either the **MTInquireJPGVals**( ) or one of the decompression functions that decompresses from a JPG file.

- `CommentLength`

Specifies the length of the information in bytes. If the contents is longer than **CommentLength**, the extra part will be dropped. The user must ensure that the **CommentLength** is not longer than the size of the **Comments** buffer.

**OutFile**

A pointer to a string that specifies the output file name which may contain a path and must have an extension of JPG.

RETURN VALUE

OK

The function completed successfully.

PARA_ERROR

One of the parameters is invalid.

COMMENTS

The Data Frame structure must be filled out accordingly before passed to this function. The image data is stored in the frame buffer one line adjacent to another without gaps. The length of a line in bytes depends on the **PixelFormat**. 8 bit Black/White format takes 1 byte per pixel, 16 bit YCbCr422 format takes 2 bytes per pixel. The total length of a line is the number of pixels per line (**dx**) times the number of bytes per pixel.

If the user wants to compress an image stored in RGB color format, the image data must be converted to YCbCr422 format by using the **MTRGB24BitToYCbCr422** function provided in the Miscllanous Utility Group before passing it to this function.

If only part of the image stored in the Data Frame is to be compressed, use the **region** to define the ROI.

The **UpDownFlip** flag is used to control if the first line to be compressed is from the top or bottom of the ROI. This is useful when the user need to invert the image while it is compressed. See the example code for **MTRGB24BitToYCbCr422**( ) function for how to use this flag.

For lossy (baseline) compression, the Quality Factor is used to control the compressed image quality. The higher the Quality Factor, the better the compressed image quality and the lower the compression ratio, and vice versa. To adjust the compression ratio and compressed image quality, MuTech recommends using 128 as the start value. Usually this value will generate acceptable results for both compression ratio and compressed image quality. The user may experiment with different settings and choose the best one.

For lossless compression, a Predictor Number is used. For details about the definition of the Predictor Number, refer to Appendix C.

SIDE EFFECT

None.

SEE ALSO

```
JPEG_File2Memory_Decompress( ),
MTInquireJPGVals( ),
MTRGB24BitToYCbCr422( )
```

EXAMPLE

```
CompressParameter cPara;
DataFrame dFrame;
```

```
ROI roi;
BYTE monoInfo, bpp;

// fill out the input data structure

// the following code segment is equivalent to
// dFrame.PixelFormat = DATA_AS_COMPRESSED
MTInquireJPGVals(“test.jpg”,
              &dFrame.dx,
              &dFrame.dy,
              &colorInfo,
              NULL, NULL);
if (monoInfo == 0)
{
    dFrame.PixelFormat = DATA_16bit_YCbCr422;
    bpp = 2;
}
else
{
    dFrame.PixelFormat = DATA_8bit_BW;
        bpp = 1;
}
// allocate memory
dFrame.buffer = (BYTE HUGE *)MTMALLOC((ULONG)dFrame.dx
                     * (ULONG)dFrame.dy
                     * (ULONG)bpp
                     );
dFrame.UpDownFlip = 0;

// fill out the region of interest structure
// here, the ROI is the same as the Data Frame
roi.sx = 0;
roi.sy = 0;
roi.dx = dFrame.dx;
roi.dy = dFrame.dy;

// decompress
if (JPEG_File2Memory_Decompress(“test.jpg”,
                   dFrame,
                   roi, NULL, NULL) != OK)
{
    // error occurs
}
.
.
// the decompressed data can be processed here
// fill out the input parameter structure
cPara.CompressMethod = JPEG_LOSSY;
cPara.QulityFactor = 128;
cPara.CompressedDataFormat = JPEG_AS_SOURCE;
cPara.Comments = NULL;
cPara.cLength = 0;

// compress
```

```
if (JPEG_Memory2File_Compress(dFrame,
                      roi,
                      cPara,
                      “newtest.jpg”) != OK)
{
    // error occurs
}
MTFREE(dFrame.buffer);
```

## 3.1.8 JPEG_File2Memory_Decompress

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL JPEG_File2Memory_Decompress(
                  BYTE FAR *InFile,
                  DataFrame dataFrame,
                  ROI region,
                  BYTE FAR *Comments,
                  UWORD FAR *cLength);
```

DESCRIPTION

Decompresses an image file in JPG format into a region of interest within a Data Frame.

PARAMETERS

**InFile**

A pointer to a string that specifies the input file name which may contain a path and must have an extension of JPG.

**dataFrame**

Specifies an image in system memory as below:

```
typedef struct DATAFRAME {
       BYTE HUGE *buffer;
       UWORD dx;
       UWORD dy;
       BYTE UpDownFlip;
       BYTE PixelFormat;
       BYTE Reserved;
       } DataFrame;
```

The members of the structure are explained below:

- buffer

  Point to a byte buffer that contains the image data. The image size and format are specified by the other members of this structure. The buffer length must be calculated as:

```
if (PixelFormat == DATA_8bit_BW) bp = 1;
else if (PixelFormat == DATA_16bit_YCbCr422 ||
       PixelFormat == DATA_16bit_YONLY)
       bp = 2;
length = dx * dy * bp;
```

- `dx`

  Specifies the number of pixels per line in the Data Frame.

- `dy`

  Specifies the number of lines in the Data Frame.

- `UpDownFlip`

  This flag specifies the direction of compressing, inverse vertically if set to 1. See the example code for **MTRGB24BitToYCbCr422**( ) function on how to use this flag.

- `PixelFormat`

  Specifies the pixel format for the Data Frame. Only the following formats are supported:

  *`DATA_8bit_BW`*
  8 bit per pixel Black/White format.

  *`DATA_16bit_YCbCr422`*
  16 bit per pixel YCbCr422 format.

  *`DATA_16bit_YONLY`*
  16 bit per pixel YCbCr422 format. But only the Y component is relavent in this case, the Cb and Cr component will be filled with values of 0x80.

  *`DATA_AS_COMPRESSED`*
  Specifies that the Decompressed Data Format format is determined by the Compressed Data Format in the input file. When the Compressed Data Format is 8 bit Black/White, the Decompressed Data Format will be DATA_8bit_BW. When the Compressed Data Format is 16 bit YCbCr422, the Decompressed Data Format will be DATA_16bit_YCbCr422.

**region**

Specifies a Region Of Interest by the **ROI** structure:

```
typedef struct ROI {
        UWORD sx;
        UWORD sy;
        UWORD dx;
        UWORD dy;
    } ROI;
```

The members of the structure are explained below:

- `sx, sy`

  Specify, in term of pixels, the offset of the upper left corner of the region in a Data Frame.

- `dx, dy`

Specify, in term of pixels and lines, the width and height of the region in a Data Frame.

Note, a region must be assoicated with a Data Frame in order to be completely defined. The **region** is used to put the decompressed image into an ROI of a larger Data Frame. To decompress the image into entire Data Frame, define the region equal to the frame. If the ROI is smaller than the image size, only the upper left part which is equal to **region** will be filled into the Data Frame.

⇦ **Comments**

Pointer to a byte buffer that will be used to hold the JPEG comment information if one exists in the compressed code stream. A NULL pointer could be used if the user has no interest of the comment content at the time of calling. If a buffer pointer is specified, its length must be specified by **cLength**, see below for detailed description. It is the caller's responsibility to allocate enough space for the buffer, otherwise the extra part of the **Comments** field will be discarded. The user can also retrieve the comment information by calling **MTInquireJPGVals**( ) function.

⇦ **cLength**

Pointer to an UWORD variable that can be used in a number of ways described below:

- points to NULL - the user is not interested in any information in the comment field. The function will not look for Comment Marker and totally ignore the **Comments** buffer.
- points to a variable with the value 0 - The user is interested in only the length of the comment field. If there is no Comment Marker found, the return value will be 0 (unchanged). Otherwise, this function will return the actual length of the comment field in the variable pointed to. The **Comments** buffer will be ignored.
- points to a variable with non-zero value - The user is interested in both the length and the content of the comment field. In this case, the **Comments** buffer must be allocated and supplied, and the value of the variable must be equal to the size of the **Comments** buffer. If there is no Comment Marker found, the return value will be 0. Otherwise, this function will copy the comment field into the buffer pointed by **Comments**, and return the actual length of the comment field in the variable pointed to.

RETURN VALUE

OK

The function completed successfully.

PARA_ERROR

One of the parameters is invalid.

COMMENTS

The Data Frame structure, except the buffer, must be filled out accordingly before passed to this function. The decompressed image data will be stored in the frame **buffer** one line adjacent to another without gaps. The length of a line in bytes depends on **PixelFormat**. 8 bit Black/White format takes 1 byte per pixel, 16 bit YCbCr422 format takes 2 bytes per pixel. The total length of a line is the number of pixels per line (**dx**) times the number of bytes per pixel.

How to determine the **PixelFormat** migth be tricky because the result will also depends on the Compressed Data Format in the input file. The easiest way is to use **DATA_AS_COMPRESSED**. It will determine the **PixelFormat** based on the Compressed Data Format in the input file. See parameter description above for **DATA_AS_COMPRESSED**. This function has some ability to convert data before writting to the **buffer**. For example, assuming the Compressed Data Format is YCbCr422 (color image). If the user specifies the **PixelFormat** as **DATA_8bit_BW**, the Cb and Cr components will be dropped and the image will be packed to 8 bit Black/White format. If the user specifies the **PixelFormat** as **DATA_16bit_YONLY**, the Cb(U) and Cr(V) components will be replaced with 0x80. Other cases are not supported.

If the user wants to have a color image in RGB format, the image data must be converted by calling the **MTYCbCr422ToRGB24Bit**( ) function provided in the Miscellanous Utility Group.

In General, there two ways that the application can retrieve the comment field from a compressed code stream. The first is to call **MTInquireJPGVals**( ) function to find out the information about the comment field, then allocate the buffer of the right size and retrieve the comment field during the decompression function call. The second way is to blindly allocate a buffer which is large enough to hold the maximum anticipated size of the comment field, then calls this function to get the information needed.

SIDE EFFECT

None.

SEE ALSO

```
JPEG_Memory2File_Compress( ),
MTInquireJPGVals( ),
MTRGB24BitYCbCr422( ),
MTYCbCr422ToRGB24Bit( )
```

EXAMPLE

See sample for **JPEG_Memory2File_Compress**( ).

## 3.1.9 JPEG_Memory2Memory_Compress

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL JPEG_Memory2Memory_Compress(
                  DataFrame dataFrame,
                  ROI region,
                  CompressParameter cmpPara,
                  BYTE HUGE *jpegBuffer,
                  ULONG FAR *bufferLength);
```

DESCRIPTION

Compresses an image in a pre-defined Data Frame in system memory into a memory buffer in JPG file format.

PARAMETERS **dataFrame**

Specifies an image in system memory as below:

```
typedef struct DATAFRAME {
       BYTE HUGE *buffer;
       UWORD dx;
       UWORD dy;
       BYTE UpDownFlip;
       BYTE PixelFormat;
       BYTE Reserved;
       } DataFrame;
```

The members of the structure are explained below:

- `buffer`

  Point to a byte buffer that contains the image data. The image size and format are specified by the other members of this structure. The buffer length must be calculated as:

  ```
  if (PixelFormat == DATA_8bit_BW) bp = 1;
  else if (PixelFormat == DATA_16bit_YCbCr422 ||
         PixelFormat == DATA_16bit_YONLY)
         bp = 2;
  length = dx * dy * bp;
  ```

- `dx`

  Specifies the number of pixels per line in the Data Frame.

- `dy`

  Specifies the number of lines in the Data Frame.

- `UpDownFlip`

  This flag specifies the direction of compressing, inverse vertically if set to 1.

- `PixelFormat`

  Specifies the pixel format for the Data Frame. Only the following formats are supported:

  *DATA_8bit_BW*
  8 bit per pixel Black/White format.

  *DATA_16bit_YCbCr422*
  16 bit per pixel YCbCr422 format.

  *DATA_16bit_YONLY*
  16 bit per pixel YCbCr422 format. But only the Y component is relavent in this case, the Cb and Cr component are treated as dummies.

**region**

Specifies a Region Of Interest by the **ROI** structure:

```
typedef struct ROI {
       UWORD sx;
       UWORD sy;
       UWORD dx;
       UWORD dy;
   } ROI;
```

The members of the structure are explained below:

- `sx, sy`

  Specify, in term of pixels, the offset of the upper left corner of the region in a Data Frame.

- `dx, dy`

  Specify, in term of pixels and lines, the width and height of the region in a Data Frame.

Note, a region must be assoicated with a Data Frame in order to be completely defined. Only the image in the **region** will be compressed. To compress the entire Data Frame, define the region equal to the frame.

**cmpPara**

Specifies the compression parameters in the **CompressParameter** structure defined below:

```
typedef struct COMPRESSPARAMETER {
       BYTE CompressMethod;
       BYTE QualityFactor;
       BYTE CompressedDataFormat;
       BYTE FAR *Comments;
       UWORD CommentLength;
       } CompressParamter;
```

The member of the structure are explained below:

- `CompressMethod`

  Specifies the compression method which can be one of the following pre-defined macros:

  *`JPEG_LOSSY`*
  Uses the JPEG Baseline Compression.

  *`JPEG_LOSSLESS`*
  Uses the JPEG Lossless Compression.

- `QualityFactor`

  Specifies a Quality Factor, which ranges from 1 to 255, when JPEG_LOSSY is used.

  When JPEG_LOSSLESS is used. it selects one of the predictors, to be used in compression. The valid value is from 1 to 7. Foe details about predictor, refer to Appendix C.

- `CompressedDataFormat`

  Specifies the Compressed Data Format with one of the following pre-defined macros:

  *`JPEG_8bit_BW`*
  Specifies that the Compressed Data Format will be 8 bit per pixel Black/White format regardless what the Pixel Format is selected in the Data Frame. If the Pixel Format in the Data Frame is DATA_16bit_YCbCr422 or DATA_16bit_YONLY, the image data will be converted to DATA_8bit_BW before compression.

  *`JPEG_16bit_YCbCr422`*
  Specifies that the Compressed Data Format will be 16 bit per pixel YCbCr422 format. This parameter is valid only when the input data format is DATA_16bit_YCbCr422.

  *`JPEG_AS_SOURCE`*
  The Compressed Data Format will be determined by input data, i.e., Pixel Format in the Data Frame. When the Pixel Format is DATA_8bit_BW, the Compressed Data Format will be JPEG_8bit_BW. When the Pixel Format is DATA_16bit_YCbCr422 or DATA_16bit_YONLY, the Compressed Data Format will be JPEG_16bit_YCbCr422.

- `Comments`

  Pointer to a byte buffer that contains application related information, such as comment, time stamp, etc. The contents in the buffer will be copied to the compressed **jpegBuffer** using a JPEG Comment Marker. The user can retrieve this information by calling either the **MTInquireJPGInMemoryVals**( ) or one of the decompression functions that decompresses from a JPG memory buffer.

- `CommentLength`

  Specifies the length of the information in bytes. If the contents is longer than **CommentLength**, the extra part will be dropped. The user must ensure that the **CommentLength** is not longer than the size of the **Comments** buffer.

⇦ **jpegBuffer**
A pointer to a buffer of bytes which will be used to store the compressed code stream in JPG format.

**bufferLength**
A ULONG variable that specifies the length of the buffer. The requirement of the buffer length depends on the compression ratio which is, in turn, depending on the Quality Factor parameter. It is the caller’s responsibility to allocate enough space to accommodating the code stream. A rule of thumb is to allocate a buffer the same size of the original image.

RETURN VALUE

OK
The function completes successfully.

PARA_ERROR
One of the parameter is invalid.

COMMENTS

This function is functionally identical to **JPEG_Memory2File_Compress**( ). The only difference is that this function stores the compressed data into a system memory buffer instead of a disk file. The data generated by these two functions are identical. Please refer to **JPEG_Memory2File_Compress**( ) for comments.

SIDE EFFECT

None.

SEE ALSO

```
JPEG_Memory2Memory_Decompress( ),
MTInquireJPGInMemoryVals( ),
JPEG_Memory2File_Compress( )
```

EXAMPLE

```
CompressParameter cPara;
DataFrame dFrame;
BYTE monoInfo, bp;
BYTE HUGE *jpegBuffer;
ULONG length;

dFrame.buffer = (BYTE HUGE *)MTMALLOC((ULONG)dFrame.dx
                     * (ULONG)dFrame.dy
                     * (ULONG)monoInfo
                      );
dFrame.UpDownFlip = 0;

// fill the data frame with 8 bit black/white data
// from other source
dFrame.PixelFormat = DATA_8bit_BW;
.
.
// fill out the region of interest structure
roi.sx = 0;
roi.sy = 0;
roi.dx = dFrame.dx;
roi.dy = dFrame.dy;

// fill out the input parameter structure
cPara.CompressMethod = JPEG_LOSSY;
cPara.QulityFactor = 128;
cPara.CompressedDataFormat = JPEG_AS_SOURCE;
cPara.Comments = NULL;
cPara.cLength = 0;

// allocate buffer to hold compressed data
// (in JPG format)
// assume the compression ratio is greater than 1:1
// the following buffer length is considered most
// conservative that assume nothing is compressed
length = (ULONG)dFrame.dx * (ULONG)dFrame.dy;

// compress
```

```
if (JPEG_Memory2Memory_Compress(dFrame,
                      roi,
                      cPara,
                      jpegBuffer,
                      &length) != OK)
{
   // error occurs
}
.
.  // now, the length contains the actual size of
.  // code stream in bytes
.
// decompress
// fill out the input data structure
MTInquireJPGInMemoryVals(jpegBuffer,
                   &dFrame.dx,
                   &dFrame.dy,
                   &colorInfo,
                   NULL, NULL);
if (monoInfo == 0)
{
   dFrame.PixelFormat = DATA_16bit_YCbCr422;
   bp = 2;
}
else
{
   dFrame.PixelFormat = DATA_8bit_BW;
   bp = 1;
}

if (JPEG_Memory2Memory_Decompress(jpegBuffer,
                      length,
                      roi,
                      dFrame,
                      NULL, NULL) != OK)
{
   // error occurs
}
.
.   // the decompressed data can be processed here
.
MTFREE(dFrame.buffer);
MTFREE(jpegBuffer);
```

## 3.1.10 JPEG_Memory2Memory_Decompress

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL JPEG_Memory2Memory_Decompress(
                  BYTE HUGE *jpegBuffer,
                  ULONG bufferLength,
                  DataFrame dataFrame,
                  ROI region,
                  BYTE FAR *Comments,
                  UWORD FAR *cLength);
```

DESCRIPTION

Decompresses a JPG format image stored in a memory buffer into a pre-defined Data Frame in system memory.

PARAMETERS

**jpegBuffer**

A pointer to a BYTE buffer that holds the compressed code stream in JPG format.

**bufferLength**

Specifies the length of the buffer. The **bufferLength** defines the byte number of code in the buffer. The **bufferLength** usually has a value returned from previous call to **JPEG_Memory2Memory_Compress**( ) function.

**dataFrame**

Specifies an image in system memory as below:

```
typedef struct DATAFRAME {
       BYTE HUGE *buffer;
       UWORD dx;
       UWORD dy;
       BYTE UpDownFlip;
       BYTE PixelFormat;
       BYTE Reserved;
       } DataFrame;
```

The members of the structure are explained below:

- `buffer`

  Point to a byte buffer that contains the image data. The image size and format are specified by the other members of this structure. The buffer length must be calculated as:

```
if (PixelFormat == DATA_8bit_BW) bp = 1;
else if (PixelFormat == DATA_16bit_YCbCr422 ||
        PixelFormat == DATA_16bit_YONLY)
        bp = 2;
length = dx * dy * bp;
```

- `dx`

  Specifies the number of pixels per line in the Data Frame.

- `dy`

  Specifies the number of lines in the Data Frame.

- `UpDownFlip`

  This flag specifies the direction of compressing, inverse vertically if set to 1. See the example code for **MTRGB24BitYCbCr422**( ) function on how to use this flag.

- `PixelFormat`

  Specifies the pixel format for the Data Frame. Only the following formats are supported:

  *`DATA_8bit_BW`*
  8 bit per pixel Black/White format.

  *`DATA_16bit_YCbCr422`*
  16 bit per pixel YCbCr422 format.

  *`DATA_16bit_YONLY`*
  16 bit per pixel YCbCr422 format. But only the Y component is relavent in this case, the Cb and Cr component will be filled with values of 0x80.

  *`DATA_AS_COMPRESSED`*
  Specifies that the Decompressed Data Format format is determined by the Compressed Data Format in the input file. When the Compressed Data Format is 8 bit Black/White, the Decompressed Data Format will be DATA_8bit_BW. When the Compressed Data Format is 16 bit YCbCr422, the Decompressed Data Format will be DATA_16bit_YCbCr.

**region**
Specifies a Region Of Interest by the **ROI** structure:

```
typedef struct ROI {
       UWORD sx;
       UWORD sy;
       UWORD dx;
       UWORD dy;
    } ROI;
```

The members of the structure are explained below:

- `sx, sy`

  Specify, in term of pixels, the offset of the upper left corner of the region in a Data Frame.

- `dx, dy`

  Specify, in term of pixels and lines, the width and height of the region in a Data Frame.

Note, a region must be assoicated with a Data Frame in order to be completely defined. The **region** is used to put the decompressed image into an ROI of a larger Data Frame. To decompress the image into entire Data Frame, define the region equal to the frame. If the ROI is smaller than the image size, only the upper left part which is equal to **region** will be filled into the Data Frame.

⇦ **Comments**

Pointer to a byte buffer that will be used to hold the JPEG comment information if one exists in the compressed code stream. A NULL pointer could be used if the user has no interest of the comment content at the time of calling. If a buffer pointer is specified, its length must be specified by **cLength**, see below for detailed description. It is the caller's responsibility to allocate enough space for the buffer, otherwise the extra part of the **Comments** field will be discarded. The user can also retrieve the comment information by **MTInquireJPGVals**( ) function.

⇦ **cLength**

Pointer to an UWORD variable that can be used in a number of ways described below:

- points to NULL **-** the user is not interested in any information in the comment field. The function will not look for Comment Marker and totally ignore the **Comments** buffer.
- points to a variable with the value 0 **-** The user is interested in only the length of the comment field. If there is no Comment Marker found, the return value will be 0 (unchanged). Otherwise, this function will return the actual length of the comment field in the variable pointed to. The **Comments** buffer will be ignored.
- points to a variable with non-zero value **-** The user is interested in both the length and the content of the comment field. In this case, the **Comments** buffer must be allocated and supplied, and the value of the variable must be equal to the size of the **Comments** buffer. If there is no Comment Marker found, the return value will be 0. Otherwise, this function will copy the comment field into the buffer pointed by **Comments**, and return the actual length of the comment field in the variable pointed to.

RETURN VALUE

OK

The function completes successfully.

PARA_ERROR

One of the parameter is invalid.

COMMENTS

This function is functionally identical to **JPEG_File2Memory_Decompress**( ). The only difference is that this function reads the compressed data from a system buffer instead of a disk file. The files generated by these two functions are identical. Please refer to **JPEG_File2Memory_Deompress**( ) for comments.

In General, there two ways that the application can retrieve the comment field from a compressed code stream. The first is to call **MTInquireJPGInMemoryVals**( ) function to find out the information about the comment field, then allocate the buffer of the right size and retrieve the comment field during the decompression function call. The second way is to blindly allocate a buffer which is large enough to hold the maximum anticipated size of the comment field, then calls this function to get the information needed.

SIDE EFFECT None.

SEE ALSO
```
JPEG_Memory2Memory_Compress( ),
MTInquireJPGInMemoryVals( ),
JPEG_File2Memory_Decompress( )
```

EXAMPLE See sample for **JPEG_Memory2Memory_Compress**( ).

## 3.2 MuTech Products Support Group

### 3.2.1 IV-4XX Board Support Group

The functions in this group must be used with IV-4XX SDK. The function prototypes for this group can be found in IV4.H provided by IV-4XX SDK. Both MuTech JPEG DLLs and corresponding IV-4XX SDK DLLs must be present for these functions to be called successfully. For functions described in this sub-section, please also refer to the document “IV-4XX Software Development Guide”.

**Note** **For functions in this group, unless specified otherwise, all coordinates and dimensions are in unites of pilxes and lines.**

#### 3.2.1.1 IV4JPEGLoad

SYNOPSIS
```
#include “iv4.h”
int MTPROCALL IV4JPEGLoad(char FAR *name,
                  UWORD sx, UWORD sy,
                  UWORD dsx, UWORD dsy,
                  UWORD dx, UWORD dy,
                  UWORD flag,
                  BYTE FAR *Comments,
                  UWORD FAR *cLength);
```

DESCRIPTION Decompresses a sub-area of the image in a JPG file and loads into an ROI in a pre-defined IV-4XX Video Frame.

PARAMETERS

**name**

specifies the file name. The file name can contain a path and must have an extension of JPG.

**sx, sy**

specifies the start point (offset) of the sub-area relative to the upper left corner of the image, horizontally (**sx**) and vertically (**sy**) in the **name**d JPG file (source).

**dsx, dsy**

specifies the start point (offset) of the ROI, horizontally (**dsx**) and vertically (**dsy**) in the pre-defined IV-4XX Video Frame (destination).

**dx, dy**

specifies the size of the sub-area. If no clipping happened, the sub- area sizes for the source and the destination should be the same.

When **sx** + **dx** is greater than the width of the compressed image, a clipping will occur in the source side, which produces a new **dx**’. When **dsx** + **dx**’ is greater than the width of the IV-4XX Video Frame, a clipping will occur in the destination side. The **sy**, **dy**, and **dsy** are working the same way on the vertical direction.

**flag**

specifies how the image should be loaded. It could be one of pre- defined macros:

*`IV4_Load_As_Is`*
The image should be loaded as is. If the image is in 8 bit format, it will be loaded in Y only Black/White format, if the image is in color format, it will be loaded in 16 bit YCbCr422 color image.

*`IV4_Load_As_BW`*
The image will be loaded in Y only Black/White format no matter what the compressed image is.

If necessary, the color space conversion is applied automatically. This is detailed in the following table.

| **flag** | Compressed Format | IV-4XX On-Board Format |
|---|---|---|
| **IV4_Load_As_Is** | Black/White | 8 bit B/W (Cb=Cr=0x80) |
| **IV4_Load_As_Is** | Color | Color, 16 bit YCbCr422 |
| **IV4_Load_As_BW** | Black/White or Color | 8 bit B/W (Cb=Cr=0x80) |

⇦ **Comments**

Pointer to a byte buffer that will be used to hold the JPEG comment information if one exists in the compressed code stream. A NULL pointer could be used if the user has no interest of the comment content at the time of calling. If a buffer pointer is specified, its length must be specified by **cLength**, see below for detailed description. It is the caller's responsibility to allocate enough space for the buffer, otherwise the extra part of the **Comments** field will be discarded. The user can also retrieve the comment information by calling **MTInquireJPGVals**( ) function.

⇦ **cLength**

Pointer to an UWORD variable that can be used in a number of ways described below:

- points to NULL - the user is not interested in any information in the comment field. The function will not look for Comment Marker and totally ignore the **Comments** buffer.
- points to a variable with the value 0 - The user is interested in only the length of the comment field. If there is no Comment Marker found, the return value will be 0 (unchanged). Otherwise, this function will return the actual length of the comment field in the variable pointed to. The **Comments** buffer will be ignored.
- points to a variable with non-zero value - The user is interested in both the length and the content of the comment field. In this case, the **Comments** buffer must be allocated and supplied, and the value of the variable must be equal to the size of the **Comments** buffer. If there is no Comment Marker found, the return value will be 0. Otherwise, this function will copy the comment field into the buffer pointed by **Comments**, and return the actual length of the comment field in the variable pointed to.

RETURN VALUE

IV4_OK

The function completed successfully.

IV4_ERROR

The board has not been initialized.

IV4_NOT_SUPPORTED

The IV-4XX SDK could not find the MuTech JPEG SDK DLL file. This error code is for Windows (3.1x, 9x, NT) only.

IV4_PARAM_ERR

Indicates one of the following cases:

- **sx** > the width of the image in the file
- **sy** > the height of the image in the file
- **dsx** > the width of the Video Frame
- **dsy** > the height of the Video Frame

- **name** does not have an extension of JPG
- invalid **flag**.

COMMENTS

This function requires the application to define a IV-4XX Video Frame by calling **IV4SetVideoFrame**( ) before loading the image. The parameter **dsx**, **dsy**, and **dx**, **dy** are referenced to the Video Frame.

The application can call **IV4InquireJPEGVals**( ) to get the image size and the data format about the compressed image and then set the IV-4XX Video Frame accordingly.

In General, there two ways that the application can retrieve the comment field from a compressed code stream. The first is to call **MTInquireJPGVals**( ) function to find out the information about the comment field, then allocate the buffer of the right size and retrieve the comment field during the decompression function call. The second way is to blindly allocate a buffer which is large enough to hold the maximum anticipated size of the comment field, then calls this function to get the information needed.

SEE ALSO

`IV4SetVideoFrame( ), IV4InquireJPEGVals( )`

EXAMPLE

```
UWORD VF_dx, VF_dy, fmt;
.
.
// this example loads an image from a JPG file
// to IV-4XX board and display it.
// Assume the iV-4XX board has been opened and
// initialized successfully.

// inquire the file information
IV4InquireJPEGVals(“image.jpg”, &VF_dx, &VF_dy, &fmt,
                     NULL, NULL);

// set the Video Frame
IV4SetVideoFrame(IV4_Single_Image_Mode,
                     NULL, VF_dx, VF_dy);

// the following call will load an image from file
// to the Video Frame and display at (0, 0)
IV4GetVideoFrame(NULL, NULL, &VF_dx, &VF_dy);
IV4SetImageFrameOnVideoFrame(0, 0, VF_dx, VF_dy);
IV4SetImageWndOnScreen(0, 0);
IV4JPEGLoad(“image.jpg”, 0, 0, 0, 0,
              VF_dx, VF_dy, IV4_Load_As_Is,
              NULL, NULL);
```

## 3.2.1.2 IV4JPEGSave

SYNOPSIS

```
#include “iv4.h”
int MTPROCALL IV4JPEGSave(char FAR *name,
                  UWORD sx, UWORD sy,
                  UWORD dx, UWORD dy,
                  UWORD flag, BYTE CompressMethod,
                  BYTE QualityFactor
                  BYTE FAR *Comments,
                  UWORD cLength);
```

DESCRIPTION

Compresses an ROI in a pre-defined IV-4XX Video Frame and saves into a JPG file.

PARAMETERS

**name**

specifies the file name. The file name can contain a path and must have an extension of JPG.

**sx, sy**

specifies the start point (offset) of the ROI, horizontally (**sx**) and vertically (**sy**) in the pre-defined IV-4XX Video Frame (source).

**dx, dy**

specifies the size of the ROI. When **sx** + **dx** is greater than the width of pre-defined IV-4XX Video Frame, a clipping will occur. The **dx** (or **dx**’ if clipping happened) is the size of the compressed image. The **dy** is working the same way on vertical direction.

**flag**

specifies how the image should be saved, it could be one of the pre-defined macros:

*IV4_Save_As_Is*
The image should be saved as is. If theimage is in B/W mode, it will be compressed in Black/White format. Color image will be in YCbCr422 format.

*IV4_Save_As_BW*
The image should be compressed in Black/White format, no matter what the original image format is.

*IV4_Save_As_YUV*
The image should be compressed in YCbCr422 format, no matter what the original image format is.

If necessary, the color space conversion is applied automatically. This is detailed in the following table.

| **flag** | IV-4XX On-Board Format | Compressed Format |
|---|---|---|
| **IV4_Save_As_Is** | B/W (Cb=Cr=0x80) | 8 bit Black/White |
| **IV4_Save_As_Is** | Color,<br>16 bit YCbCr422 | 16 bit YCbCr422 |
| **IV4_Save_As_BW** | B/W (Cb=Cr=0x80) or Color (16 bit YCbCr422) | 8 bit B/W |
| **IV4_Save_As_YUV** | B/W (Cb=Cr=0x80) or Color (16 bit YCbCr422) | 16 bit YCbCr422 |

**CompressMethod**

Specifies the compression method which can be one of the following pre-defined macros:

*IV4_JPEG_LOSSY*
Uses the JPEG Baseline Compression.

*IV4_JPEG_LOSSLESS*
Uses the JPEG Lossless Compression.

**QualityFactor**

Specifies a Quality Factor, which ranges from 1 to 255, when IV4_JPEG_LOSSY is used.

When IV4_JPEG_LOSSLESS is used. it selects one of the predictors, to be used in compression. The valid value is from 1 to 7.

**Comments**

Pointer to a byte buffer that contains application related information, such as comment, time stamp, etc. The contents in the buffer will be copied to the compressed file using a JPEG Comment Marker. The user can retrieve this information by calling either the **MTInquireJPGVals**( ) or the **IV4JPEGLoad**().

**cLength**

Specifies the length of the information in bytes. If the contents is longer than **cLength**, the extra part will be dropped. The user must ensure that the **cLength** is not longer than the size of the **Comments** buffer.

RETURN VALUE

IV4_OK
The function completed successfully.

IV4_ERROR
The board has not been initialized.

IV4_NOT_SUPPORTED

The IV-4XX SDK could not find the MuTech JPEG SDK DLL file. This error code is for Windows (3.1x, 9x, NT) only.

IV4_PARAM_ERR

Indicates one of the following cases:

- **sx** > the width of the IV-4XX Video Frame
- **sy** > the height of the IV-4XX Video Frame
- **name** does not have an extension of JPG
- invalid **flag**.

COMMENTS

For lossy (baseline) compression, Quality Factor is used to control the compressed image quality. The higher the Quality Factor, the better the compressed image quality and the lower the compression ratio, and vice versa. To adjust the compression ratio and compressed image quality, MuTech recommends using 128 as the Quality Factor to start value. Usually this value will generate acceptable results for both compression ratio and compressed image quality. The user may experiment with different settings and choose the best one.

For lossless compression, a Predictor Number is used. For details about the definition for the Predictor Number, refer to Appendix C.

SEE ALSO

`IV4SetVideoFrame( ), IV4JPEGLoad( )`

EXAMPLE

```
UWORD DefaultSx, DefaultSy;
// this example grab an single frame into the
// Video Frame and compress and save to a .JPG file
.
// open and initialize the board
.
.
// set to use NTSC camera
IV4SetTVStandard(IV4_Camera_NTSC);

// the default sx, sy may vary among different
// TV standards and also among different cameras
DefaultSx = 0x0033;
DefaultSy = 0x000A;

// set grab window to grab frame with odd field
// as first field
IV4SetGrabWindow(IV4_Frame_Mode | IV4_Grab_Odd,
            DefaultSx, DefaultSy,
            IV4_NTSC_Width_Default,// 640
            IV4_NTSC_Height_Default);// 480
// set the Video Frame
IV4SetVideoFrame(IV4_Single_Image, 0, 0,
            IV4_NTSC_Width_Default,// 640
            IV4_NTSC_Height_Default);// 480
```

```
// grab one frame into the Video Frame
IV4StartGrab(IV4_Single_Grab);

// the following call will save an image from
// the Video Frame to a .JPG file
IV4JPEGSave("image.jpg", 0, 0,
        IV4_NTSC_Width_Default,// 640
        IV4_NTSC_Height_Default,// 480
        IV4_Save_As_Is,
        IV4_JPEG_LOSSY,
        128, NULL, 0);
.
.
```

## 3.2.1.3 IV4InquireJPEGVals

SYNOPSIS

```
#include "iv4.h"
int MTPROCALL IV4InquireJPEGVals(char *name,
                  UWORD *dx, UWORD *dy,
                  UWORD *fmt,
                  BYTE FAR *Comments,
                  UWORD FAR *cLength);
```

DESCRIPTION

Inquires the image size and format in a specified JPG file.

PARAMETERS

**name**

specifies the file name. The file name can contain a path and must have an extension of JPG.

⇦ **dx, dy**

pointer to UWORD variables that return the size of the image in terms of pixels and lines in the JPG file.

⇦ **fmt**

pointer to an UWORD variable that returns the color depth of the image in JPG file. The possible values returned can be 8, or 16 (bits).

⇦ **Comments**

Pointer to a byte buffer that will be used to hold the JPEG comment information if one exists in the compressed code stream. A NULL pointer could be used if the user has no interest of the comment content at the time of calling. If a buffer pointer is specified, its length must be specified by **cLength**, see below for detailed description. It is the caller's responsibility to allocate enough space for the buffer, otherwise the extra part of the **Comments** field will be discarded. The user can also retrieve the comment information by calling **MTInquireJPGVals**( ) function.

⇦ **cLength**

Pointer to an UWORD variable that can be used in a number of ways described below:

- points to NULL **-** the user is not interested in any information in the comment field. The function will not look for Comment Marker and totally ignore the **Comments** buffer.
- points to a variable with the value 0 **-** The user is interested in only the length of the comment field. If there is no Comment Marker found, the return value will be 0 (unchanged). Otherwise, this function will return the actual length of the comment field in the variable pointed to. The **Comments** buffer will be ignored.
- points to a variable with non-zero value **-** The user is interested in both the length and the content of the comment field. In this case, the **Comments** buffer must be allocated and supplied, and the value of the variable must be equal to the size of the **Comments** buffer. If there is no Comment Marker found, the return value will be 0. Otherwise, this function will copy the comment field into the buffer pointed by **Comments**, and return the actual length of the comment field in the variable pointed to.

RETURN VALUE

IV4_OK
The function completed successfully.

IV4_ERROR
The board has not been initialized.

IV4_PARAM_ERR
**name** does not have an extension name of JPG,

COMMENTS

The application can call this function to get the image size and the data format about the JPG file and then set the Video Frame accordingly before calling to **IV4JPEGLoad**( ).

In general, there two ways that the application can retrieve the comment field from a compressed code stream. The first is to call this function to find out the information about the comment field, then allocate the buffer of the right size and retrieve the comment field during the **IV4JPEGLoad**( ) function call. The second way is to blindly allocate a buffer which is large enough to hold the maximum anticipated size of the comment field, then calls this function to get the information needed. When one of the decompression function is called, using the NULL pointer option to disregard the comment field.

SEE ALSO

`IV4SetVideoFrame( ), IV4JPEGLoad( )`

EXAMPLE

See the example for **IV4JPEGLoad**( ).

## 3.3 Miscllaneous Utility Group

Functions in this group provide a variety of color space conversion or file information inquiry capability.

### 3.3.1 MTRGB24BitToYCbCr422

SYNOPSIS

```
#include “mtjpeg.h”
void MTPROCALL MTRGB24BitToYCbCr422(
            void HUGE *input, void HUGE *output,
            ULONG length, UWORD RGBOrder,
            UWORD SubSample);
```

DESCRIPTION

Converts a stream of RGB data to a stream of YCbCr422 (YUYV) data.

PARAMETERS

**input**

Pointer to the input buffer which contains the RGB data to be converted. The length of the input buffer must be at least 3 times the number of pixels (**length** * 3). The data sequence of the input buffer is defined by the **RGBOrder** parmeter.

⇦ **output**

Pointer to the output buffer which is used to store the resulte of the conversion. The length of the output buffer must be at least 2 times the number of pixels (**length** * 2). The data format of the output buffer is YCbCr422 (YUYV), where each unit corresponds to each pair of RGB pixels.

**length**

Specifies the number of pixels to be converted. The length must be even number.

**RGBOrder**

Specifies the order of the RGB data strored in the **input** buffer. When **RGBOrder** is 1, the input data format is RGBRGB..., when **RGBOrder** is 0, the input data format is BGRBGR....

**SubSample**

Reserved and must be set to 0.

RETURN VALUE

None

COMMENTS

Also see comments for **JPEG_Memory2File_Compress**( ).

SEE ALSO

```
MTYCbCr422ToRGB24Bit( )
JPEG_Memory2File_Compress( )
```

## 3.3.2 MTYCbCr422ToRGB24Bit

SYNOPSIS

```
#include “mtjpeg.h”
void MTPROCALL MTYCbCr422ToRGB24Bit(
            void HUGE *input, void HUGE *output,
            ULONG length, UWORD RGBOrder,
            UWORD SubSample);
```

DESCRIPTION

Converts a stream of YCbCr data to a stream of RGB data.

PARAMETERS

**input**

Pointer to the input buffer which contains the YCbCr data to be converted. The length of the input buffer must be at least 2 times the number of pixels (**length** * 2). The data is YCbCr422 (YUYV), where each unit corresponds to each pair of RGB pixels.

⇦ **output**

Pointer to the output buffer which is used to store the results of the conversion.The length of the output buffer must be at least 3 times the number of pixels (**length** * 3). The data sequence of the output buffer is defined by the **RGBOrder** parmeter.

**length**

Specifies the number of pixels to be converted. The length must be even number.

**RGBOrder**

Specifies the order of the RGB data strored in the **output** buffer. When **RGBOrder** is 1, the output data format is RGBRGB..., when **RGBOrder** is 0, the output data format is BGRBGR....

**SubSample**

Reserved and must be set to 0.

RETURN VALUE

None

COMMENTS

Also see comments for **JPEG_File2Memory_Decompress**( )

SEE ALSO

```
MTRGB24BitToYCbCr422( ),
JPEG_File2Memory_Decompress( )
```

EXAMPLE

See example for **MTRGB24BitToYCbCr422**( ).

## 3.3.3 MTRGB32BitToYCbCr422

SYNOPSIS

```
#include “mtjpeg.h”
void MTPROCALL MTRGB32BitToYCbCr422(
            void HUGE *input, void HUGE *output,
            ULONG length, UWORD RGBOrder,
            UWORD SubSample);
```

DESCRIPTION

Converts a stream of RGB data to a stream of YCbCr422 (YUYV) data.

PARAMETERS

**input**

Pointer to the input buffer which contains RGB data to be converted. The length of the input buffer must be at least 4 times the number of pixels (**length** * 4). The data sequence of the input buffer is defined by the **RGBOrder** parmeter.

⇦ **output**

Pointer to the output buffer which is used to store YCbCr. The length of the output buffer must be at least 2 times the number of pixels (**length** * 2). The data format of the output buffer is YCbCr422 (YUYV), where each unit corresponds to each pair of RGB pixels.

**length**

Specifies the number of pixels to be converted. The length must be even number.

**RGBOrder**

Specifies the order of the RGB data strored in the **input** buffer. When **RGBOrder** is 1, the input data format is RGBXRGBX..., when **RGBOrder** is 0, the input data format is BGRXBGRX..., where X is the byte discarded in the conversion.

**SubSample**

Reserved and must be set to 0.

RETURN VALUE

None

COMMENTS

Also see comments for **JPEG_Memory2File_Compress**( ).

SEE ALSO

`MTYCbCr422RGB32Bit( ), JPEG_Memory2File_Compress( )`

## 3.3.4 MTYCbCr422ToRGB32Bit

SYNOPSIS

```
#include “mtjpeg.h”
void MTPROCALL MTYCbCr422ToRGB32Bit(
            void HUGE *input, void HUGE *output,
            ULONG length, UWORD RGBOrder,
            UWORD SubSample);
```

DESCRIPTION

Converts a stream of YCbCr data to a stream of RGB data.

PARAMETERS

**input**

Pointer to the input buffer which contains YCbCr data to be converted. The length of the input buffer must be at least 2 times the number of pixels (**length** * 2). The data format of the input buffer is YCbCr422 (YUYV), where each unit corresponds to each pair of RGB pixels.

⇦ **output**

Pointer to the output buffer which is used to store the results of the conversion.The length of the output buffer must be at least 4 times the number of pixels (**length** * 4). The data sequence of the output buffer is defined by the **RGBOrder** parmeter.

**length**

Specifies the number of pixels to be converted. The length must be even number.

**RGBOrder**

Specifies the order of the RGB data strored in the output buffer. When **RGBOrder** is 1, the output data format is RGBXRGBX..., when **RGBOrder** is 0, the output data format is BGRXBGRX..., where X is the byte discarded in the conversion.

**SubSample**

Reserved and must be set to 0.

RETURN VALUE

None

COMMENTS

Also see comments for **JPEG_File2Memory_Decompress**( )

SEE ALSO

`MTRGB32BitToYCbCr422( ), JPEG_File2Memory_Decompress( )`

## 3.3.5 MTInquireJPGVals

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL MTInquireJPGVals(char FAR *file,
            UWORD FAR *dx, UWORD FAR *dy,
            UWORD FAR *fmt,
            BYTE FAR *Comments, UWORD FAR *cLength);
```

DESCRIPTION

Inquires the image size and format in a specified JPG file.

PARAMETERS

**file**

Specifies the file name. The file name can contain a path and must have an extension of JPG.

⇦ **dx, dy**

Pointer to UWORD variables that return the size of the image in terms of pixels and lines in the JPG file.

⇦ **fmt**

Pointer to an UWORD variable that returns the image type. The possible values returned are 0 for color image (YCbCr422), or 1 for Black/White image (8 bit B/W).

⇦ **Comments**

Pointer to a byte buffer that will be used to hold the JPEG comment information if one exists in the compressed code stream. A NULL pointer could be used if the user has no interest of the comment content at the time of calling. If a buffer pointer is specified, its length must be specified by **cLength**, see below for detailed description. It is the caller’s responsibility to allocate enough space for the buffer, otherwise the extra part of the Comments field will be discarded. The user can also retrieve the comment information by using any one of the decompression functions that decompresses from a JPG file.

⇦ **cLength**

Pointer to an UWORD variable that can be used in a number of ways described below:

- points to NULL - the user is not interested in any information in the comment field. The function will not look for Comment Marker and totally ignore the Comments buffer.
- points to a variable with the value 0 - The user is interested in only the length of the comment field. If there is no Comment Marker found, the return value will be 0 (unchanged). Otherwise, this function will return the actual length of the comment field in the variable pointed to. The Comments buffer will be ignored.

- points to a variable with non-zero value - The user is interested in both the length and the content of the comment field. In this case, the Comments buffer must be allocated and supplied, and the value of the variable must be equal to the size of the Comments buffer. If there is no Comment Marker found, the return value will be 0. Otherwise, this function will copy the comment field into the buffer pointed by Comments, and return the actual length of the comment field in the variable pointed to.

RETURN VALUE

OK
The function completed successfully.

PARAM_ERR
file does not have an extension of JPG,

COMMENTS

The application calls this function to get the image size and the data format of the JPG file. It is helpful that the application knows the information before further operates on the file.

In general, there two ways that the application can retrieve the comment field from a compressed code stream. The first is to call this function to find out the information about the comment field, then allocate the buffer of the right size and retrieve the comment field during the decompression function call. The second way is to blindly allocate a buffer which is large enough to hold the maximum anticipated size of the comment field, then calls this function to get the information needed. When one of the decompression function is called, using the NULL pointer option to disregard the comment field.

SEE ALSO

`JPEG_Memory2File_Compress( )`

EXAMPLE

See the example for **JPEG_Memory2File_Compress**( ).

## 3.3.6 MTInquireJPGInMemoryVals

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL MTInquireJPGInMemoryVals(
      BYTE HUGE *jpegBuffer, ULONG bufferLength,
      UWORD FAR *dx, UWORD FAR *dy, UWORD FAR *fmt,
      BYTE FAR *Comments, UWORD FAR *cLength);
```

DESCRIPTION

Inquires the image size and format of compressed data in a specified memory buffer in JPG file format.

PARAMETERS

**jpegBuffer**
Pointer to a BYTE buffer that will be used to store the compressed data in JPG format.

**bufferLength**

Specifies the length of the **jpegBuffer**. The bufferLength defines the byte number of the code in the buffer. It usually has a value returned from provious call to a comprssion function that is using the memory buffer to hold the compressed code stream.

⇦ **dx, dy**

Pointer to UWORD variables that return the size of the image in in terms of pixels and lines in the memory buffer in JPG format.

⇦ **fmt**

Pointer to an UWORD variable that returns the image type. The possible values returned are 0 for color image (YCbCr422), or 1 for Black/White image (8 bit B/W).

⇦ **Comments**

Pointer to a byte buffer that will be used to hold the JPEG comment information if one exists in the compressed code stream. A NULL pointer could be used if the user has no interest of the comment content at the time of calling. If a buffer pointer is specified, its length must be specified by **cLength**, see below for detailed description. It is the caller's responsibility to allocate enough space for the buffer, otherwise the extra part of the Comments field will be discarded. The user can also retrieve the comment information by using any one of the decompression functions that decompresses from a memory buffer in JPG file format.

⇦ **cLength**

Pointer to an UWORD variable that can be used in a number of ways described below:

- points to NULL - the user is not interested in any information in the comment field. The function will not look for Comment Marker and totally ignore the Comments buffer.
- points to a variable with the value 0 - The user is interested in only the length of the comment field. If there is no Comment Marker found, the return value will be 0 (unchanged). Otherwise, this function will return the actual length of the comment field in the variable pointed to. The Comments buffer will be ignored.
- points to a variable with non-zero value - The user is interested in both the length and the content of the comment field. In this case, the Comments buffer must be allocated and supplied, and the value of the variable must be equal to the size of the Comments buffer. If there is no Comment Marker found, the return value will be 0. Otherwise, this function will copy the comment field into the buffer pointed by Comments, and return the actual length of the comment field in the variable pointed to.

RETURN VALUE

OK

The function completed successfully.

PARAM_ERR
**jpegBuffer** is NULL.

COMMENTS

The application calls this function to get the image size and the data format in a memory buffer in JPG file format. It is helpful that the application knows the information before further operates on the memory buffer.

In General, there two ways that the application can retrieve the comment field from a compressed code stream. The first is to call this function to find out the information about the comment field, then allocate the buffer of the right size and retrieve the comment field during the decompression function call. The second way is to blindly allocate a buffer which is large enough to hold the maximum anticipated size of the comment field, then calls this function to get the information needed. When one of the decompression function is called, using the NULL pointer option to disregard the comment field.

SEE ALSO

```
JPEG_Memory2Memory_Compress( ),
JPEG_File2Memory_Compress( )
```

EXAMPLE

See the example for **JPEG_Memory2Memory_Compress**( ).

## 3.3.7 MTInquireBMPVals

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL MTInquireBMPVals(BYTE FAR *file,
            UWORD FAR *dx, UWORD FAR *dy,
            UWORD FAR *fmt);
```

DESCRIPTION

Inquires the image size and format in a specified BMP file.

PARAMETERS

**file**
Specifies the file name. The file name can contain a path and must have an extension of BMP.

⇦ **dx, dy**
Pointer to UWORD variables that return the size of the image in terms of pixels and lines in the BMP file.

⇦ **fmt**
Pointer to an UWORD variable that returns the color depth of the image in the BMP file. The possible values returned are 8, or 24 (bits).

RETURN VALUE

OK
The function completed successfully.

PARAM_ERR
**file** does not have an extension of BMP,

COMMENTS

The application can call this function to get the image size and the data format about the BMP file. It is helpful that the application need to knows the information before further operates on that file.

SEE ALSO

`JPEG_File2Memory_Compress( )`

EXAMPLE

See the example for **JPEG_File2Memory_Compress**( ).

## 3.3.8 MTInquireTGAVals

SYNOPSIS

```
#include “mtjpeg.h”
int MTPROCALL MTInquireTGAVals(BYTE FAR *file,
            UWORD FAR *dx, UWORD FAR *dy,
            UWORD FAR *fmt);
```

DESCRIPTION

Inquires the image size and format in a specified TGA file.

PARAMETERS

**file**
Specifies the file name. The file name can contain a path and must have an extension of TGA.

⇦ **dx, dy**
Pointer to UWORD variables that return the size of the image in terms of pixels and lines in the TGA file.

⇦ **fmt**
Pointer to an UWORD variable that returns the color depth of the image in the TGA file. The possible values returned are 8, or 24 (bits).

RETURN VALUE

OK
The function completed successfully.

PARAM_ERR
**file** does not have an extension name of TGA,

COMMENTS

The application can call this function to get the image size and the data format about the TGA file. It is helpful that the application knows the information before further operates on that file.

SEE ALSO

`JPEG_File2Memory_Compress( )`

EXAMPLE

See the example for **JPEG_File2Memory_Compress**( ).

## 3.3.9 MTInquireTIFFVals

SYNOPSIS

```
#include "mtjpeg.h"
int MTPROCALL MTInquireTIFFVals(BYTE FAR *file,
            UWORD FAR *dx, UWORD FAR *dy,
            UWORD FAR *fmt);
```

DESCRIPTION

Inquires the image size and format in a specified TIF file.

PARAMETERS

**file**

Specifies the file name. The file name can contain a path and must have an extension of TIF.

⇦ **dx, dy**

Pointer to UWORD variables that return the size of the image in terms of pixels and lines in the TIF file.

⇦ **fmt**

Pointer to an UWORD variable that returns the color depth of the image in the TIF file. The possible values returned are 8, 16, or 24 (bits).

RETURN VALUE

OK

The function completed successfully.

PARAM_ERR

**file** does not have an extension of TIF,

COMMENTS

The application can call this function to get the image size and the data format about the TIF file. It is helpful that the application knows the information before further operates on that file.

SEE ALSO

`JPEG_File2Memory_Compress( )`

EXAMPLE

See the example for **JPEG_File2Memory_Compress**( ).

# Appendix A: Glossary

| | |
|---|---|
| compression | Reduction in the number of bits used to represent source image data. |
| compression data | Either compressed image data or table specification or both. |
| DCT coefficient | The amplitude of the specific cosine basis function. |
| decode | A process which takes the input compressed image data and outputs a continuous tone image. |
| dequantization | The inverse procedure to quantization by which the decoding process recovers a representation of the DCT coefficients. |
| discrete cosinetransform; DCT | Either the forward discrete cosine transform or the inverse discrete cosine transform. |
| Huffman decoding | An entropy decoding procedure which recovers the symbol from each variable length code produced by the Huffman encoder. |
| Huffman encoding | An entropy encoding procedure which assigns a variable length code to each input symbol. |
| Huffman table | The set of variable length codes required for Huffman encoding and Huffman decoding. |
| JPEG | The Joint Photographic Experts Group. This group is a joint ISO/CCITT technical committee (ISO/IEC JTC1/SC2/WG10, Photographic Image Coding) whose goal has been to develop a general-purpose international standard for gray scale or true color compression. |
| lossless | A term for encoding and decoding processes and procedures in which the output of the decoding procedure(s) is identical to the input to the encoding procedure(s). |
| lossy | A term for encoding and decoding processes which are not lossless. |

| | |
|---|---|
| predictor | A linear combination of previously encoded reconstructed values (uses in loss-less mode coding). |
| quantization | The procedure by which the DCT coefficients are linearly scaled in order to achieve better compression. |
| (8x8)block | An 8x8 array of samples. |
| SubSampling | A procedure by which the spatial resolution of an image is reduced. |

# Appendix B: Overview of the JPEG Still Picture Compression Algorithm

By Gregory K. Wallace
Digital Equipment Corporation
Maynard, Massachusetts

## B.1 Introduction

For the past few years, a standardization effort known by the acronym JPEG, for Joint Photographic Experts Group, has been working toward establishing the first international digital image compression standard for between CCITT and ISO, JPEG convenes officially as the ISO committee with nomenclature JTC1/SC2/WG10, but operates in close collaboration with CCITT SGVIII.

The only widely used international digital image compression standard in existence today is the bi-level compression method used by Group 3 and Group 4 facsimile machines. This method differs from the methods being standardized by the JPEG committee in a number of important ways. For one, the JPEG methods are applicable only to multi-level images and not at all to bi-level images--just the opposite of the existing fax methods. Moreover, the JPEG standard aims to be a general-purpose technique for applications as diverse as photo-videotex, desktop publishing, graphic arts, color facsimile, newspaper wire photo transmission, medical systems, and many others. The Group 3/4 compression methods, however, were standardized expressly for fax machines (though they are being adopted for other applications as well).

As was the case for commonplace facsimile before the Group 3 standards were established, each of these continuous-tone imaging applications needs its own image compression standard in order to flourish, so that equipment from different manufacturers will interoperate. But the JPEG committee has a strong additional conviction that most of these applications can be satisfied by a common general-purpose image compression standard. This would facilitate exchange of images across application boundaries as these applications become increasingly internetworked. Moreover, state-of-the-art methods for continuous-tone image compression often require VLSI implementation to achieve the coding or decoding speed required by many applications--a common general-purpose method would promise to reduce significantly the cost of specialized compression hardware.

The price of this aim for application independence is a proposed standard which has been fairly long in the making, and which on first blush appears somewhat complex by virtue of its multiple modes of operation. It does, however, feature a core mode known as the Baseline System, which is common to all other modes of operation, and which is expected to be sufficient in its own right for a number of applications.

## B.2 Algorithm Requirements and Selection Process

The JPEG committee's goal has been to develop a method for continuous-tone image compression which meets the following requirements:

(a) Be at or near state of the art with regard to compression rate and accompanying image fidelity, over a wide range of image quality ratings, and especially in the range where visual fidelity to the original is characterized as "very good" to "excellent" to indistinguishable".

(b) Be applicable to practically any kind of scene content--not be limited, for example, to classes of imagery with restrictions on scene complexity, range of colors, statistical properties, etc.;

(c) Have tractable computational complexity (with some preference given to the decoding side), to yield software implementations with viable performance on a range of CPUs, as well as hardware implementation with viable cost for applications requiring high performance;

(d) Have the following modes of operation:

- Sequential build-up; each component of the image is encoded in a single left-to-right, top-to-bottom scan;
- Progressive build-up; the image is encoded in multiple scans for applications in which transmission time is long, and the user prefers to watch the image build up in multiple coarse to clear passes;
- Hierarchical encoding: the image is encoded at multiple resolutions, so that lower-resolution versions may be accessed without having first to decompress the image's full resolution.

- Lossless compression: an encoding which will guarantee exact recovery of every source image pixel value, in spite of relatively poor (compared to lossy with excellent fidelity) compression rate.

In June 1987, JPEG conducted a selection process based on a blind assessment of subjective picture quality, and narrowed 12 proposed algorithms to three. Three informal working groups formed to refine them, and in January 1988, a second, similar selection process revealed that the “ADCT” proposal, based on the 8x8 Discrete Cosine Transform (DCT), had produced the best picture quality. Since then, an intensive, group-wide effort to refine, test, and document the DCT-based algorithm, in all its modes of operation, has been underway. Details of this selection and evaluation process, as well has history of the JPEG committee, are contained in {1,2,3,4}.

The requirement for a lossless mode of operation proved to be the most difficult one to satisfy with a DCT based algorithm. For some time, JPEG considered the approach whereby the encoder would reconstruct the same compressed data sent to the decoder, form the difference of the reconstructed and source images, and send the difference image to the decoder. But this method does not guarantee a lossless result unless both encoder and decoder use identical Inverse DCT (DCT) implementations.

The JPEG committee was reluctant to specify some unique IDCT as a requirement in its proposed standard, because research into new fast-DCT transformed algorithm continues, and no one algorithm is optimal for all implementations. Instead the committee chose to specify a simple predictive method, which is entirely separate and independent from the DCT based algorithm. This method uses a 3-pixel predictor, with a lossless encoding of the prediction error.

## B.3 Basic DCT Block Diagram

Common to all encoding modes of the DCT-based algorithm are the components shown in Figure B-1, which illustrates the key processing steps applied to each 8x8 block of samples from a single component of a source image. (The encoder must extend the right and bottom edges of the image component, so that it will consist of an integral number of blocks. The actual dimensions of the image are encoded in the header, and the decoder in turn discards the extensions).

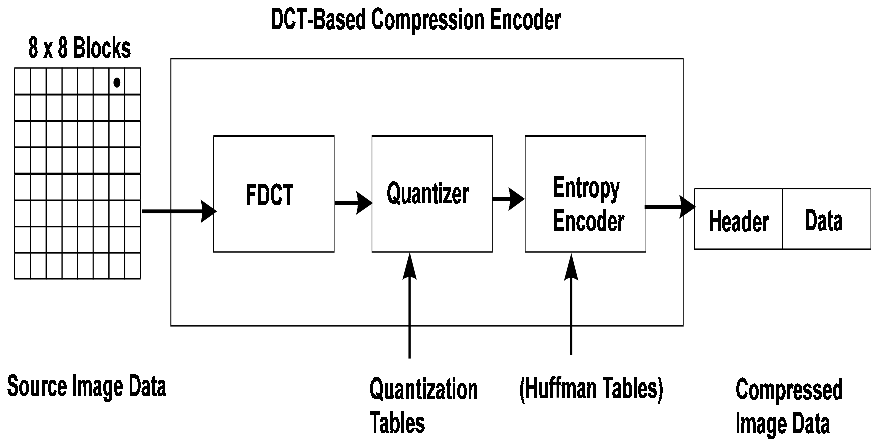

**Figure 2-1: DCT Based Encoder Functional Blocks**

The 64 image samples are the first transformed into 64 DCT coefficients by the Forward DCT (FDCT). The DCT is a relative of the Discrete Fourier Transform, and the FDCT may be regarded intuitively as a harmonic analyzer which decomposes each 8x8 block of pixels into a set of 64 two-dimensional (2D) spatial frequencies, each with a vertical and horizontal component. The coefficient values may then be regarded as a measure of the amount of each 2D frequency present in the 8x8 pixel block. The coefficient with zero frequency in both dimensions is known as the DC coefficient and the other 63 are known as the AC coefficients.

Each of the 64 DCT coefficients is next uniformly quantized in conjunction with a 64-element Quantization Table (Q-Table), which must be specified as an input to the encoder. Each element may be specified as any integer value from 1 to 255, and represents the quantizer step size for the corresponding DCT coefficient. Quantization is performed by dividing each DCT coefficient by its corresponding Q-Table element and rounding to the nearest integer. Ideally, each stepsize should be chosen by the application as the perceptual threshold for the visual contribution of its corresponding cosine basis function, as a function of the intended source image format, display characteristics, and viewing distance.

The degree of precision required to perform these computations depends on the range of the input pixels. A property of the 8x8 FDCT is that the non-fractional part of the DCT coefficients has a range which is a factor of 8 greater than that of the input pixels. Given that the minimum quantizer step size is one integer level, the quantized DCT coefficients increase in size by three bits compared to the input pixels, and greater precision than this is required to compute the FDCT itself with adequate accuracy. The JPEG committee has chosen to allow either 8 bit or 12 bit per component input pixel samples, knowing that the higher precision will require higher cost computation, but will accommodate additional applications.

After quantization, the DC coefficient is treated separately from the 63 AC coefficients. Because there is typically strong correlation between the DC components of adjacent 8x8 blocks, the quantized DC term is represented by encoding its difference from the DC term of the previous block. The special treatment is worthwhile, as the DC terms typically contain a significant fraction of the total image energy.

Entropy coding is the final processing step. Whereas compression is achieved in the quantization step by discarding data which is not visually significant, entropy coding achieves additional compression without additional loss, by encoding the data more compactly based on its statistical characteristics. Except for the Baseline System, in which only Huffman coding is allowed, either Huffman coding or Arithmetic coding may be used as the entropy coding method for all DCT-based modes of operation.

Huffman coding {4} requires use of one or more sets of Huffman code tables. A set consists of one table for the DC coefficients and one table for the AC coefficients. Each image component may use only one set of Huffman tables, though not all components must use the same set. Huffman tables may be predefined and used by an application as defaults, or computed specifically for the image in an initial statistics-gathering pass. Such choices are the business of specific applications, and outside the domain of the JPEG standard which specifies no default Huffman (or quantization) tables.

In contrast, Arithmetic coding {5} requires no tables to be externally input, because it adapts to the image statistics as it encodes the image in a single pass. Furthermore, Arithmetic coding has produced 5-10% better compression than Huffman for many of the images which the JPEG committee has tested. However, it is somewhat more complex than Huffman coding, and may be less feasible than Huffman for the highest speed hardware implementations.

For color images with moderately complex scenes, all DCT-based modes of operation typically produce the following levels of picture quality for the indicated ranges of compression. These levels are only a guideline - quality and compression can vary dramatically according to source image characteristics and scene content:

- 0.25-0.5 bits/pixel: moderate to good quality, sufficient for some applications;
- 0.5-0.75 bits/.pixel: good to very good quality, sufficient for many applications;
- 0.75-1.5 bits/pixel: excellent quality, sufficient for most applications.
- 1.5-2.0 bits/pixel: usually indistinguishable from the originals, sufficient for the most demanding applications.

The lossless procedures tend to produce about 2:1 compression for color images with moderately complex scenes.

## B.4 Source Images and Data Interleaving

The previous section described how a single component of an image is decomposed into 8x8 blocks and then transformed, quantized, and coded for the DCT-based modes of operation. While some images ("grayscale" or "monochrome" images) consist only of a single component, the JPEG standard is designed to handle also the more general class of multiple-component images, which includes all varieties of color images, and the other types as well.

A multiple-component image maybe compressed in either interleaved or non-interleaved order. In the latter case, each image component is compressed in its entirety and output as part of the compressed datastream before starting the next component. In the interleaved case, 8x8 blocks from each component are processed in round-robin fashion, and compressed blocks are output in interleaved order into the compressed stream. The case of a three-component image being encoded in interleaved order is shown in Figure B-2. (For simplicity, only one set of tables is shown.) Only one table can be assigned to each component, so the encoder must switch to the appropriate table as it switches between source image components. Whether interleaved or non-interleaved, all DCT-based algorithms encode each image component independently of all others.

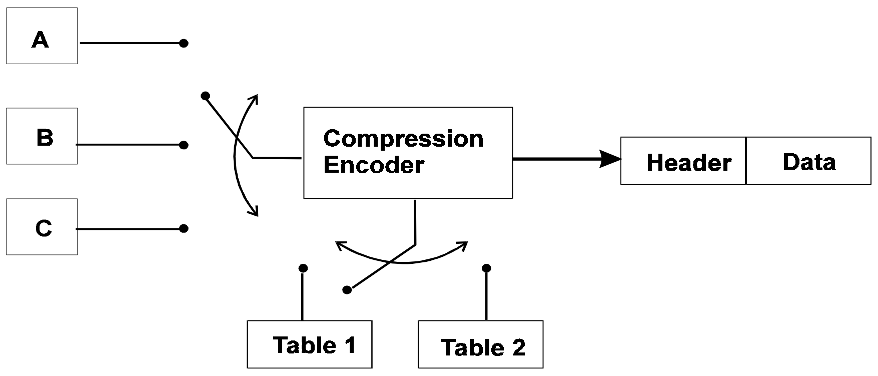

**Figure 2-1: Componen Interleave and Table-Switching**

Because the JPEG standard handles multiple-component images in this general way, it is applicable to images represented in practically any color system. Indeed, there exist many different color systems, new ones are still being invented in current research, and many are the subject of ongoing standardization efforts which are independent of JPEG's work. Consequently, the compressed image header in the JPEG standard does not contain any parameter which indicates the color space of the compressed image. To JPEG, such information is specific to the application, and not required to decompress the image.

The source image may also have one or more components that are subsampled with respect to its others, though there are restrictions on this flexibility.

## B.5 Sequential Build-up and Baseline System

Sequential build-up refers to the encoding mode in which 8x8 pixel blocks are compressed and output from left to right and top to bottom, in a single scan per component. If the image components may be interleaved, in order to limited to four each the number of Q-tables and sets of Huffman tables (or, if Arithmetic coding is employed, the number of statistics areas) which must be stored in the decoder.

Baseline System {4} is the name given to a restricted version of the sequential build-u-mode of DCT compression. It is required to be present in all DCT-based modes of operation, in order to provide a “fall-back” or default mode for applications like facsimile, in which sender and receiver can negotiate a set of common features prior to image encoding and transmission. But, the Baseline System is also expected to be sufficient for many applications in its own right. Its restrictions with respect to general sequential operation are:

(a) Operates on images with 8-bits/pixels/component only;

(b) Uses Huffman coding only;

(c) Uses at most two sets of Huffman tables per scan.

These restrictions are aimed at reducing the cost of the codec: (a) reduces the computation cost of the FDCT and IDCT as described in section 3, (b) reduces the options available for entropy coding, and (c) reduces the storage cost required at the decoder.

## B.6 Progressive Build-up

The mode of operation known as progressive build-up {6} is in many ways identical to sequential build-up. The difference is that each image component is encoded as multiple scans rather than as a single scan. In this mode, a rough but recognizable version of the image appears quickly (in comparison to the total transmission time) at the viewer’s screen, and is refined by successive scans until reaching the level of picture quality that was established by the quantization tables.

Referring to Figure 1, the progressive build-up mode of operation can be understood by imagining the addition of a full image buffer memory at the output of the quantizer, before the input to the entropy coder. The buffer memory must be of sufficient size to store the image as quantized DCT coefficients, each of which is 3 bits large than the per-component input pixels.

In progressive build-up, after each block of DCT coefficients is quantized, it is stored in the coefficient buffer memory, because only a portion of its content is encoded and output at each scan through the image. This contrasts with sequential build-up, where all coefficients in a block are quantized and then immediately entropy-coded and output at their full (quantized) resolution, without buffering.

There are two independent ways by which the quantized coefficients of each block may be partially encoded within a scan. First, only a specified “band” i.e. only some of the 64 coeffiients need be encoded. This mode is called spectral selection, because each band typically contains coefficients which occupy a lower or higher part of the spatial-frequency “spectrum” for that 8x8 block. Secondly, the coefficients within the current band need not be encoded to their full (quantized) accuracy within each scan. Upon a coefficient’s first encoding, the N most significant bits can be encoded first, where N is specifiable. In one or more subsequent refinement scans, the bits of less significance can then be encoded. This second mode is called successive approximation.

## B.7 Hierarchical Encoding

The hierarchical mode {2} provides a “pyramidal” encoding of an image at multiple resolutions, each differing in resolution from its adjacent encoding by a factor of two in either the horizontal or vertical dimension or both. The encoding procedure can be summarized as follows:

(a) Filter and down-sample the original image by the desired number of multiples of 2 in each dimension.

(b) Encode this reduced-size image using either the sequential or progressive build-up methods previously described.

(c) Decode this reduced-size image and then interpolate and up-sample it by 2 horizontally and/or vertically, using the identical interpolation filter which the receiver must use.

(d) Use this up-sampled image as a prediction of the original at this resolution, and encode the difference image using either the sequential or progressive encoding modes.

(e) Repeat steps (c) and (d) until the full resolution of the image has been encoded.

Hierarchical encoding is useful in applications in which a very high resolution image must be accessed by a lower-resolution device, which does not have the buffer capacity to reconstruct the image at its full resolution and then scale it down for the lower-resolution display. An example scanned and compressed at high resolution for a very high-quality printer, where the image must also be displayed on a low-resolution PC video screen.

## B.8 Standardization Schedule

The JPEG committee has completed revision 8 {7} of its technical specification, the details of which are now quite stable. The sequential modes of operation have been validated by exchange of encoded data among committee members, to verify that independently produced codecs operate compatibly. This validation procedure is currently underway with the progressive, hierarchical, and lossless modes of operations as well. Unless an error is discovered, the technical definition is considered to be frozen by the committee.

The technical specification is now being converted to the Committee Draft (CD), which is the document in the official ISO standards format to be formally balloted. The CD ballot is expected to commence before the end of 1990. Upon successful completion of this ballot, expected before mid-1991, the Draft International Standard (DIS) document will be balloted. Successful completion of the DIS phase will result in a final International Standard, which the committee hopes to achieve before the end of 1991.

# Appendix C: Prediction

Figure C-1 shows the relationship between the positions (a, b, c) of the reconstructed neighboring samples used for prediction and the position of x, the sample being coded.

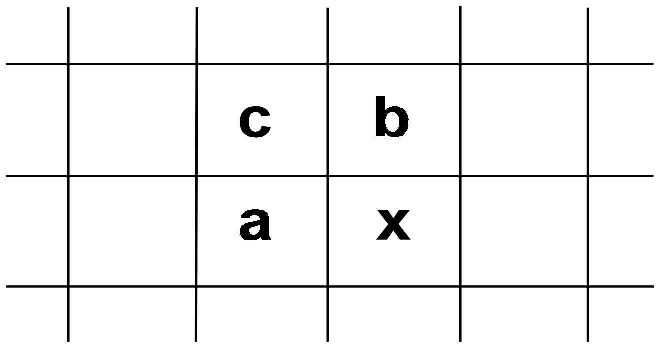

**Figure 3-1: Relationship between Sample & Prediction Samples**

Define Px to be the prediction and Ra, Rb, and Rc to be the reconstructed samples immediately to the left, immediately above, and diagonally to the left of the current sample. The allowed predictors, one of which is selected in the scan header, are listed in Table C-1.

| **Predictor Number** | **Prediction** |
|---|---|
| **0** | No prediction |
| **1** | Px = Ra |
| **2** | Px = Rb |
| **3** | Px = Rc |
| **4** | Px = Ra + Rb - Rc |
| **5** | Px = Ra + ((Rb - Rx) / 2) |
| **6** | Px = Rb + ((Ra - Rc) / 2) |
| **7** | Px = (Ra + Rb) / 2 |

**Table 0-1: Predictors for Lossless Coding**

Selections 1, 2, and 3 are one-dimensional predictors and selections 4, 5, 6, and 7 are two-dimenstional predictors The one-dimensional horizontal predictor (prediction sample Ra) is used for the first line of samples at the start of the scan and at the beginning of each restart interval. The selected predictor is used for all other lines. The sample from the line above (prediction sample Rb) is used at the start of each line, except for the first line. At the beginning of the first line and at the beginning of each restart interval the prediction value of $2^{P-1}$ is used, where P is the input precision.

# Index