Imagery Programmer's Reference

Imagery is a Windows Dynamic Linked Library designed to simplify graphics support for a host application by providing a library of graphics related functions, including file input and output and several image manipulation functions.

Features
Library Requirements

API Reference
Structures and Tables
Usage Notes

Shareware Notice
Registration Form
Credits and Copyright
Support

Features

JPEG support:
- Baseline JPEG
- CMYK JPEG
- Thumbnail in JPEG support (registered version)

TIFF support:
- CMYK TIFF
- JPEG TIFF
- Tiled TIFF files

PNG support:
- Conforms to the PNG Spec v1.0
- Read/write interlaced PNG files

other formats:
- Windows Bitmap
- Silicon Graphics
- SUN Raster
- PhotoShop 2.5
- Zsoft PCX
- Portable Pixmap
- Targa

Image Processing:
- Contrast/Brightness/Gamma alteration
- Colour space conversion (Floyd-steinberg dithering)
- Increase Noise
- Pixel scattering
- Pixel Blurring, Averaging and Sharpening
- Embossing
- Flip
- Rotate
- Resize/Resample
- Equalize/Normalize
- 5x5 Matrix operations

API Reference

InitLib
EndLib

ReadImage
WriteImage
GetFileInfo

AddNoise
AverageImage
BlurImage
CombineChannels
CropImage
DetectEdges
DisplaceImage
EmbossImage
EqualizeImage
FilterImage
FlipImage
GammaCorrect
GetChannel
InvertImage
NormalizeImage
PutChannel
ResampleImage
ResizeImage
RotateImage
SharpenImage
TuneImage

Library Requirements

Hardware:
Imagery requires an absolute minimum of a 386 processor, however Imagery functions most efficiently with a 486 or greater. Although the amount of memory required to successfully process an image depends on image size, 8 MB of RAM is required for effective functionality.

Software:
The 16-bit version requires Windows™ 3.1 or higher and a compiler which supports calling functions in a DLL. If the 32-bit DLL is used, a 32-bit compiler is required and Windows NT™ or Windows 95™ is recommended.

InitLib

Syntax: UINT InitLib(UINT *reserved*, HWND *hWnd*, UINT *wm_message*);

Purpose: Initialize user defined variables and functional state.

Parameter	Description
reserved	Reserved for future use
hWnd	Window handle of calling application
wm_message	Message to send, range from WM_USER to 7FFFH

Notes: InitLib should be called before any other function of the Imagery library is called. The ideal place to call InitLib is in the WinMain function before the main message loop, or in the instance initialization function.

InitLib is used to communicate vital instance specific information relating to the host application. The first parameter is reserved and currently has no function, the second parameter is the window handle of the calling application. The window handle provided is used to send messages to and is used to display message boxes if a fatal error occurs. The third parameter is used to give a message handle to Imagery so that status information (percentage complete) can be sent to the calling application using a Windows message. The message is usually received in the main message loop.

If the message handle is not in the range WM_USER to 7FFFH no messages are sent to the calling application (useful in Visual Basic). The current percentage of completion of the function being called is the *wParam* of the message.

If InitLib is not called before using other functions in Imagery the shareware message will be displayed every time a function in the library is called.

Return: If successful and a valid message handle was given then the return will be the given message handle. If the message handle supplied was invalid or initialization failed the return value is 0.

EndLib

Syntax: int EndLib(void);

Purpose: Perform cleanup of library state initialised in InitLib

Notes: EndLib needs no parameters, cleanup is done by setting all global settings of the library to NULL also all separate library components are de-initialised and any memory allocated in InitLib is freed.

Return: If successful EndLib returns TRUE otherwise if de-initialisation failed FALSE will be returned.

ReadImage

Syntax: HANDLE ReadImage(LPSTR *filename*);

Purpose: Reads any supported image format into a DIB handle

Parameter	Description
filename	Full pathname of the image file

Notes: ReadImage will read any supported file format into an appropriate DIB handle. To determine how to interpret the image file header is read to find any format specific features, if no such features exist in the first few bytes then ReadImage will fail.

If the image file being read is grayscale the returned DIB will be an 8-bit DIB with a grayscale palette.

Return: If successful ReadImage returns a DIB handle otherwise 0 is returned.

WriteImage

Syntax: UINT WriteImage(LPSTR *filename,* HANDLE *hDIB,* WriteOptions* *Options*);

Purpose: Write any supported image format from a DIB handle

Parameter	**Description**
filename	Full pathname of the image file
hDIB	Handle to a DIB
Options	**WriteOptions** structure containing format specific information

Notes: WriteImage uses the information provided in the **WriteOptions** structure to write the appropriate format with the desired attributes. If the values in the WriteOptions structure are inconsistent or unpractical then default values will be used.

If an 8-bit image is passed to WriteImage and the requested output file format is does not support a palette based colorspace then it is assumed the DIB is grayscale.

Return: If successful WriteImage will return TRUE, if there is an error FALSE will be returned.

Example: This Example function is a small piece of code which writes a JPEG file with 4:1:1 sampling and a quality factor of 80, followed by a TIFF file with 16 kb strips.

```
void Write_Test_File(HANDLE hDIB) /* Function Write_Test_File, write the given DIB to a JPEG and a TIFF file */
{
	WriteOptions FileOptions; /* declare structure*/
	UINT ret;

	FileOptions.FileFormat = FILE_JPEG; /* set file format option to JPEG */
	FileOptions.ImageClass = IMG_RGB_24; /* set colorspace to 24-bit RGB */
	FileOptions.JPEGSampling = SAMPLE_411; /* set sampling factors to 4:1:1 */
	FileOptions.JPEGQuality = 80; /* set jpeg quality to 80 (very high quality) */
	ret = WriteImage( "testfile.jpg", hDIB, &FileOptions ); /* write the JPEG file */
	FileOptions.FileFormat = FILE_TIFF; /* set file format to TIFF */
	FileOptions.ImageClass = IMG_RGB_24; /* set colorspace to 24-bit RGB*/
	FileOptions.SegmentSize = 16; /* set strip size to 16 kb per strip */
	FileOptions.Compression = TIFF_NONE; /* set compression to none */
	ret = WriteImage( "testfile.tif", hDIB, &FileOptions ); /* write the TIFF */
	return;
}
```

GetFileInfo

Syntax: int GetFileInfo(LPSTR *filename,* FileInfo **Image_info*);

Purpose: Fetches simple information on any supported file format

Parameter	**Description**
filename	Name of the image file
Image_info	FileInfo structure to receive JPEG information

Notes: GetFileInfo decodes the header of any supported file and fills a FileInfo structure with the information retrieved. If GetFileInfo cannot successfully decode the header of an image file then the file is not readable

Return: If successful the message handle is returned otherwise it is 0.

GetChannel

Syntax: HANDLE GetChannel(HANDLE *hDIB*, int *index*);

Purpose: Get a specified channel from a DIB

Parameter	**Description**
hDIB	DIB with multiple channels
index	0 based index of the component to retrieve

Note: GetChannel retrieves a specified channel from a 24-bit DIB. This could be used to retrive the Red channel (channel 0) from a DIB and then modify the channel without affecting other channels (ie Green and Blue).

Return: If successful a DIB handle is returned otherwise it is 0.

PutChannel

Syntax: HANDLE PutChannel(HANDLE *hDIB*, HANDLE *hDIBchannel* int *index*);

Purpose: Put a specified channel into a DIB

Parameter	**Description**
hDIB	DIB with multiple channels
hDIBchannel	DIB with single channel
index	0 based index of the component to set

Note: PutChannel may be used to put a colour channel into a DIB after the channel has been affected. The following example shows how PutChannel and GetChannel can be used to switch Red and Blue channels:

```
HANDLE SwitchChannels (HANDLE hDIB)
{
HANDLE hDIBred,hDIBblue; /* Declare local variables*/

hDIBred = GetChannel ( hDIB, 0); /* Retrieve red channel */
hDIBblue = GetChannel ( hDIB, 2); /* Retrieve blue channel */
hDIB = PutChannel ( hDIB, hDIBblue, 0); /* Replace red with blue*/
hDIBblue = GlobalFree ( hDIBblue); /* Free Blue channel */
hDIB = PutChannel ( hDIB, hDIBred, 2); /* Replace blue with red*/
hDIBred = GlobalFree ( hDIBred); /* Free Red channel */
return hDIB; /* Return DIB with switched channels*/
}
```

Return: If successful a DIB handle is returned otherwise it is 0.

CombineChannels

Syntax: HANDLE CombineChannels(HANDLE *hDIBRed*, HANDLE *hDIBGreen*, HANDLE *hDIBBlue*);

Purpose: Combine 3 grayscale channels to create an RGB DIB

Parameter	Description
hDIBRed	Handle to the Red component DIB
hDIBGreen	Handle to the Green component DIB
hDIBBlue	Handle to the Blue component DIB

Note: CombineChannels combines 3 grayscale DIBs representing Red, Green and Blue. The difference between calling CombineChannels once and PutChannel 3 times is that CombineChannels allocates the memory for the 24-bit DIB and fills the BITMAPINFOHEADER correctly, also if 3 channels are to be combined this is more efficient.

Return: If successful a DIB handle is returned otherwise it is 0.

ColorConvert

Syntax: HANDLE ColorConvert(HANDLE *hDIB*,int *imgclass*)

Purpose: Convert a DIB to a different color space

Parameter	**Description**
hDIB	Handle to the DIB
imgclass	Type of image data format and colorspace desired

Note: This function is not currently designed for speed or selection of conversion methods. Currently supported conversions are:

From

IMG_RGB_24:
- IMG_RGB_8 - 2-pass Floyd-Steinberg dithering
- IMG_GRAY_8 - basic RGB -> GRAY conversion

IMG_RGB_8:
- IMG_RGB_24 - simple upsample
- IMG_GRAY_8 - sets palette to grays

IMG_GRAY_8:
- IMG_RGB_24 - simple upsample
- IMG_RGB_8 - makes sure there is a palette associated with a DIB

Return: If successful a DIB handle is returned otherwise it is 0.

GammaCorrect

Syntax: HANDLE GammaCorrect(HANDLE *hDIB*,float *red,* float *green,* float *blue*)

Purpose: Gamma Correct an image

Parameter	**Description**
hDIB	Handle to a DIB
red	red correction value between 0.1 and 7.0
green	green correction value between 0.1 and 7.0
blue	blue correction value between 0.1 and 7.0

Note: If the gamma value is below 1 the image is darkened, if the value is above 1 the image is lightened

Return: If successful a DIB handle is returned otherwise it is 0.

AddNoise

Syntax: HANDLE AddNoise(HANDLE *hDIB*,int *amount*,int *type*)

Purpose: Add a specified amount of noise to an image

Parameter	**Description**
hDIB	Handle to a DIB
amount	amount of noise to add
type	type of noise, currently does little

Note: The higher the amount the more noise is introduced to the DIB

Return: If successful a DIB handle is returned otherwise it is 0.

CropImage

Syntax: HANDLE CropImage(HANDLE *hDIB*, RECT *CropRect*);

Purpose: Scale an image, without arbritary aspect ratios

Parameter	**Description**
hDIB	Handle to a DIB
CropRect	RECT structure defining cropping rectangle

Return: If successful a DIB handle is returned otherwise it is 0.

DisplaceImage

Syntax: HANDLE DisplaceImage(HANDLE *hDIB*,int *amount*);

Purpose: Scatter the pixels of an image by a specified amount

Parameter	**Description**
hDIB	Handle to a DIB
amount	amount to displace pixels

Note: The higher the amount the further apart the pixels are scattered

Return: If successful a DIB handle is returned otherwise it is 0.

FlipImage

Syntax: HANDLE FlipImage(HANDLE *hDIB*, int *direction*);

Purpose: Flip a DIB vertically or horizontally

Parameter	**Description**
hDIB	Handle to the DIB
direction	Direction in which to flip, 0 for horizontal, 1 for vertical

Return: If successful a DIB handle is returned otherwise it is 0.

InvertImage

Syntax: HANDLE InvertImage(HANDLE *hDIB*);

Purpose: Invert/Negative an image

Parameter	**Description**
hDIB	Handle to a DIB

Return: If successful a DIB handle is returned otherwise it is 0.

RotateImage

Syntax: HANDLE RotateImage(HANDLE *hDIB*, float *degree*);

Purpose: Rotates an image by a specified amount

Parameter	**Description**
hDIB	Handle of the DIB to be embossed
degree	degree to rotate DIB

Note: The rotation algorithm used is not currently efficient, both in speed or memory usage, this is largely due to the use of anti-aliasing to smooth the edges of the new DIB.

Return: If successful a DIB handle is returned otherwise it is 0.

ResizeImage

Syntax: HANDLE ResizeImage(HANDLE *hDIB*, int *width,* int *height*);

Purpose: Resize the width and height of a DIB

Parameter	Description
hDIB	Handle of the DIB to be resized
width	New Width of DIB
height	New Height of DIB

Note: ResizeImage is an interface to the Windows API and in the process of resizing the DIB is transferred to a DDB. This means that if the DIB is not compatible with the DC the DIB could be subjected to Windows dithering.

Return: If successful a DIB handle is returned otherwise it is 0.

EmbossImage

Syntax: HANDLE EmbossImage(HANDLE *hDIB*, BYTE *level*);

Purpose: Smooths or averages the pixels of an Image

Parameter	**Description**
hDIB	Handle of the DIB to be embossed
level	Emboss depth, (1-7)

Note: If the source DIB was RGB data then the embossed image will not be grayscaled, if the source is paletted the image is assumed to be grayscale, that is the palette is a gradient of gray values.

Return: If successful a DIB handle is returned otherwise it is 0.

NormalizeImage

Syntax: HANDLE NormalizeImage(HANDLE *hDIB*);

Purpose: Normalize the color range of the image

Parameter	Description
hDIB	Handle of the DIB be equalized

Note: NormalizeImage creates a histogram from the pixel values in *hDIB* and normalizes the range of values in the histogram. The histogram is then remapped to the DIB. The histogram creation is not reported in the progress reporting since it does not take long to create.

Return: If successful a DIB handle is returned otherwise it is 0.

ComputeImage

Syntax: HANDLE ComputeImage(HANDLE *hDIB1*,HANDLE *hDIB2*,HANDLE *hDIBalpha*,int *method*,int *blend*,BOOL *alpha*){

Purpose: Perform a computation on the pixels of an image

Parameter	**Description**
hDIB1	Handle of source DIB
hDIB2	Handle of destination DIB
hDIBalpha	Handle of alpha mask
method	Computation to be performed
blend	Percentage to blend Source and destination
alpha	Alpha mask is used if TRUE

Notes: The blend argument is only necessary is the BLEND operation is performed, it is the percentage of the first source image to blend with the second source image. The different types of computations available are:

Definition	**Computation**
C_ ADD	Add the pixel values of *hDIB1* and *hDIB2*
C_ SUBTRACT	Subtract pixel values of *hDIB2* from *hDIB1*
C_ DIFFERENCE	Find the difference between the pixel values of *hDIB1* and *hDIB2*
C_ COMPOSITE	A simple composite of *hDIB1* and *hDIB2* using the supplied mask
C_ BLEND	Blend the pixels by a specified opacity (requires blend to be in range 1..99)
C_ MULTIPLY	Multiply the pixel values together and divide the result by 255
C_ LIGHTER	Compare the pixels of both images, using only the lighter pixels
C_ DARKER	Compare the pixels of both images, using only the darker pixels

Return: If ComputeImage is successful a handle to the computed DIB is returned, otherwise 0 is returned

CompositeImage

Syntax: HANDLE CompositeImage(HANDLE *hDIB1*,HANDLE *hDIBa1*,HANDLE *hDIB2*,HANDLE *hDIBa2*,int *mask_method*, int *method*,int *opacity*,BOOL *channels[3]*, int *x*, int *y*)

Purpose: Composite two DIBs with optional alpha masks

Parameter	Description
hDIB1	Handle of source DIB
hDIBa1	Handle of source alpha mask
hDIB2	Handle of destination DIB
hDIBa2	Handle of destination alpha mask
mask_method	Computation on mask to be performed
method	Computation to be performed
opacity	Percentage to blend Source and destination
channels	TRUE for channel to be used in composite
x	X Position of composite
y	Y Position of composite

Notes: The blend argument is only necessary is the BLEND operation is performed, it is the percentage of the first source image to blend with the second source image. The different types of computations available are:

Definition	Computation
C_ ADD	Add the pixel values of the two images together
C_ SUBTRACT	Subtract pixel values from Source Image 1
C_ DIFFERENCE	Find the difference between the pixel values of the images
C_ COMPOSITE	A simple composite of images using the mask from *SImage*
C_ BLEND	Blend the pixels by a specified opacity
C_ MULTIPLY	Multiply the pixel values together and divide the result by 255
C_ LIGHTER	Compare the pixels of both images, using only the lighter pixels
C_ DARKER	Compare the pixels of both images, using only the darker pixels

Return: If CompositeImage is successful a handle to the composited DIB is returned, otherwise 0 is returned

EqualizeImage

Syntax: HANDLE EqualizeImage(HANDLE *hDIB*);

Purpose: Equalize the color range of the image

Parameter	Description
hDIB	Handle of the DIB be equalized

Note: EqualizeImage creates a histogram from the pixel values in *hDIB* and equalizes the range of values in the histogram. The histogram is then remapped to the DIB. The histogram creation is not reported in the progress reporting since it does not take long to create.

Return: If successful a DIB handle is returned otherwise it is 0.

ResampleImage

Syntax: HANDLE ResampleImage(HANDLE *hDIB,* int *new_width,* int *new_height*);

Purpose: Resize an image using a higher quality method

Parameter	**Description**
hDIB	Handle of the DIB be equalized
new_width	New width of the DIB
new_height	New height of the DIB

Note: Unlike ResizeImage ResampleImage uses an anti-aliasing resampling algorithm. This means that features of the image which might be left jaggy by ResizeImage are smoothed to look like the original image was made at the resampled size. The disadvantages of the algorithm is that it is a great deal slower than ResizeImage.

Return: If successful a DIB handle to the resampled image is returned, otherwise 0 is returned.

DetectEdges

Syntax: HANDLE DetectEdges(HANDLE *hDIB*);

Purpose: Detect Edges of a DIB

Parameter	**Description**
hDIB	Handle of the DIB to have edges detected

Note: DetectEdges uses a 3x3 matrix to 'Detect' the edges of an image.

Return: If DetectEdges is successful a DIB handle with the processed image is returned otherwise 0 is returned.

SharpenImage

Syntax: HANDLE SharpenImage(HANDLE *hDIB*, int *amount*);

Purpose: Sharpens the pixels of an image

Parameter	Description
hDIB	Handle of the DIB to be sharpened
amount	amount of sharpening (1-99)

Note: SharpenImage uses a 3x3 matrix to sharpen the pixels of an image. To see visible results the amount of sharpening should be 10 or more.

Return: If SharpenImage is successful a DIB handle to the sharpened image is returned, if sharpen image failed 0 will be returned.

BlurImage

Syntax: HANDLE BlurImage(HANDLE *hDIB*, int *amount*);

Purpose: Blurs the pixels of an Image

Parameter	**Description**
hDIB	Handle of the DIB to be blurred
amount	amount of blurring (1-99)

Note: BlurImage uses a 3x3 matrix to blur an image, increasing the amount of blurring should not significantly affect the amount of time taken to blur an image since the amount of blurring alters the weights of the matrix. To see visible effects a factor of 10 should be used.

Return: If BlurImage is successful a DIB handle to the blurred image is returned, if BlurImage failed then 0 will be returned.

AverageImage

Syntax: HANDLE AverageImage(HANDLE *hDIB*, int *amount*);

Purpose: Averages the pixels of an Image

Parameter	Description
hDIB	Handle of the DIB to be averaged
amount	amount of averaging (1-99)

Note: BlurImage uses a 5x5 matrix to average an image, increasing the amount of averaging should not significantly affect the amount of time taken to average an image since the amount of averageing alters the weights of the matrix. AverageImage is similar to BlurImage in effect except it is a lot more noticeable.

Return: If AverageImage is successful a DIB handle to the averaged image is returned, if AverageImage failed then 0 will be returned.

FilterImage

(registered version only)

Syntax: HANDLE FilterImage(HANDLE *hDIB*, int *factor,* int *bias,* int *matrix[25]*);

Purpose: Perform a transformation of image data.

Parameter	**Description**
hDIB	Handle of the DIB to be transformed
factor	factor to divise matrix convolution
bias	bias to be added to pixels
matrix	The array which holds the matrix values

Note: This function allows the use of a matrix which can be used to perform various effects on an image. A matrix can be used to sharpen, emboss, blur and detect the edges of an image. Hue, Saturation and Luminance can also be adjusted by using a matrix.
The steps taken to perform a matrix convolution are:
1. Pixel values and corresponding matrix values are multiplied together.
2. The resulting values are added together for each pixel.
3. The sum is then divided by the factor (which must not be zero).
4. The bias is added to the final pixel values.

Return: If successful a DIB handle is returned otherwise it is 0.

TuneImage

Syntax: HANDLE TuneImage(HANDLE *hDIB*, int *brightness[3],* int *contrast[3],* int *type*);

Purpose: Tune image brightness and contrast.

Parameter	**Description**
hDIB	Handle of the DIB to be transformed
brightness	Array of red,green and blue brightening percentages
contrast	Array of red,green and blue contrast percentages
type	currently unused.

Note: TuneImage 'Tunes' the brightness and contrast of an image. The brightening algorithm used in this function is more accurate and more efficient than the one used in BrightenImage, so it should be used instead of BrightenImage.

The brightness and contrast percentages must be in the range of -100 and 100. Each element in the array passed to the TuneImage is the percentage used for each color component, the array is in red, green, blue order.

If contrast of brightness must stay the same then set the unwanted parameter to NULL.

Return: If successful a DIB handle to the tuned image is returned, if TuneImage failed then 0 will be returned.

Structures and Tables

WriteOptions
FileInfo
Format Support Table
Definition Tables

WriteOptions

```
typedef struct WriteOptions{
        int ImageClass;
        BOOL bWriteAlpha;
        BOOL bWritePreview;
        int JPEGQuality;
        int Compression;
        int FileFormat;
        int SegmentSize;
        BOOL bPredictor;
        int JPEGSampling;
        LPSTR szComments;
        BOOL bInterlaced;
} WriteOptions;
```

Field	Description
ImageClass	Colorspace of destination image
bWriteAlpha	TRUE if alpha should be written (registered version only)
bWritePreview	TRUE if preview should be written (registered version only)
JPEGQuality	quality factor for JPEG, range of 5 - 95
Compression	compression method for TIFF
FileFormat	Destination file format
SegmentSize	strip size for TIFF, must be 4,8 or 16
bPredictor	TRUE if horizontal prediction should be used in LZW TIFF
JPEGSampling	sampling factors for JPEG files
lpComments	Comments to be put in the file (none if NULL)
bInterlaced	TRUE if interlacing should be used for a PNG file

FileInfo

```
typedef struct FileInfo{
        int Width;
        int Height;
        DWORD ImageSize;
        int BitsPerPixel;
        DWORD FileSize;
        int ImageClass;
        int FileFormat;
        char szComments[80];
        int CompressionRatio;
        int CompressionMethod;
        HANDLE hPreview;
        DWORD PreviewSize;
        int PreviewWidth;
        int PreviewHeight;
        BYTE Colormap[3][256];
        int JPEGQuality;
        int JPEGSampling;
        LPSTR szJPEGType;
        BOOL bInterlaced;
        int bPreview;
        int bReadable;
        int bMask;
 } FileInfo;
```

Field	Description
Width	Width in pixels of image
Height	Height in pixels of image
ImageSize	Image size in bytes of image data
BitsPerPixel	number of bits per pixel (4,8,16,24,32)
FileSize	size in bytes of image file
ImageClass	color space of image data
FileFormat	file format of image file
szComments	comments found in header
CompressionRatio	currently unused
CompressionMethod	type of compression (TIFF definitions)
hPreview	DIB with image preview if one is found
PreviewSize	size in bytes of preview image data
PreviewWidth	width in pixels of preview image
PreviewHeight	height in pixels of preview image
Colormap	colormap of image
JPEGQuality	Approximate JPEG quality factor used in encoding
JPEGSampling	Sampling factors for JPEG file
szJPEGType	Literal description of the JPEG encoding method
bInterlaced	TRUE if a PNG file is interlaced
bPreview	TRUE if there is a preview image in file
bReadable	TRUE if the file is supported and readable
bMask	TRUE if there is an alpha channel (also refferred to as a mask)

Format Support Table

This table is a listing of supported file types and the supported sub-types of various file types.

Format	ColorSpace	Compression	Imagery Limitations
SUN Raster	Palette/Gray	none,RLE	Cannot compress RLE
	RGB	none,RLE	
	RGBA	none,RLE	
Silicon Graphics	Gray	none,RLE	Cannot compress RLE
	RGB	none,RLE	
	RGBA	none,RLE	
Compuserve GIF	Palette/Gray	LZW	Restricted (legal reasons)
Windows/OS2 BMP	Palette/Gray	none	RLE BMP files unsupported
	RGB	none	
Zsoft PCX	Palette/Gray		
	RGB		
Portable Pixmap	Gray	none	Only reads binary data
	RGB	none	
Truevision Targa	Palette	none,RLE	Cannot compress RLE
	Gray	none,RLE	Cannot write 16-bit
	RGB	none,RLE	
	RGBA	none,RLE	
JPEG	Gray	JPEG	Baseline only
	RGB	JPEG	
	YCbCr	JPEG	
	CMYK	JPEG	
TIFF	Palette / Palette A	none,LZW	LZW restricted
	Gray / Gray A	none,LZW,JPEG	cannot support 1-bit,4-bit, 16-bit
	RGB / RGBA	none,LZW	no RLE/FAX support
	YCbCr / YCbCrA	none,LZW,JPEG	
	CMYK / CMYKA	none,LZW,JPEG	
PhotoShop Bitmap	Palette / Palette A	none,RLE	Cannot compress RLE
	Gray / Gray A	none,RLE	Cannot read all PhotoShop 3.0
	RGB / RGB A	none,RLE	files
	CMYK / CMYK A	none,RLE	
PNG	Palette	LZW	
	Gray / Gray A	LZW	
	RGB / RGBA	LZW	

Key:
Palette- 8-bit color space with up to 256 values, each value is an index in a color palette
Gray - 8-bit color space with 256 levels of gray.
RGB - 24-bit RGB color space with 256 levels of red, green and blue, may be BGR order
CMYK - 32-bit CMYK color space, 256 levels of cyan, magneta, yellow and black.
YCbCr- also known as YUV, Luminance and two chrominace components,used in JPEG.
A - alpha channel, referred to as a mask.

Definition Tables

File Definition Table
TIFF Compression Table
Colorspace Table
Sampling Factors Table

File Definition Table

File Type	Value	Description
FILE_BMP	0	Windows style BMP file
FILE_OS2	1	OS/2 style BMP file
FILE_GIF	2	Compuserve GIF (unsupported)
FILE_PPM	3	Portable Pixmap
FILE_TGA	5	Truevision TARGA
FILE_TIFF	6	Aldus TIFF
FILE_PSD	7	Adobe PhotoShop Bitmap
FILE_PCX	8	Zsoft PCX
FILE_SGI	9	Silicon Graphics image file
FILE_SUNRAS	11	SUN Raster file
FILE_PNG	12	Portable Network Graphics format
FILE_JPEG	13	JPEG

TIFF Compression Table

TIFF compression Type	Value	Description
TIFF_NONE	0	No compression
TIFF_LZW	1	LZW style compression (unsupported)
TIFF_PACK	2	PackBits compression
TIFF_JPEG	8	JPEG compression

Colorspace Table

Colorspace	Value	Description
IMG_RGB_24	0	24 bpp, BGR/RGB order
IMG_RGB_8	2	8 bpp, paletted
IMG_GRAY_8	5	8 bpp, gray palette/ no palette
IMG_CMYK	6	32 bpp, CMYK
IMG_YCbCr	8	24 bpp

Sampling Factors Table

JPEG sampling factors	Value	Description
SAMPLE_NONE	0	1:1:1 sampling, RGB/YCbCr/grayscale
SAMPLE_411	1	4:1:1 sampling, RGB/YCbCr
SAMPLE_422	2	4:2:2 sampling, RGB/YCbCr
SAMPLE_2112	3	2:1:1:2 sampling, CMYK
SAMPLE_1111	4	1:1:1:1 sampling, CMYK

Usage Notes

To use Imagery in a C/C++ compiler include imagery.h in the program and link with imagery.lib, or use the LoadLibrary function to load the DLL dynamically. Visual Basic currently has no equivalent BAS file for imagery.h, so before calling a function from Visual Basic the appropriate declarations must be made,

the same applies for Pascal compilers and Delphi and other similar products. There is work on a BAS file for Visual Basic.

It should be noted that for functions which accept a DIB as a parameter will return a modified version of the DIB, this means where possible Imagery will try to modify the pixels of the DIB passed to it, rather than create a new DIB. Some functions however will change the size of the image data, requiring a new DIB to be created. The following example shows this:

```
hDIB=InvertImage(hDIB);
```

This will not lose any memory, provided InvertImage is successful.

The function definitions used by Imagery refer to Windows™ definitions. These definitions may be found in windows.h and other related files.

Shareware Notice

Imagery is distributed as shareware software. Imagery may be evaluated without charge for 30 days, after the evaluation period Imagery must be registered if it is to be used further. The shareware version cannot be distributed with any software, shareware or otherwise without prior permission. If Imagery is used to develop an application for profit then Imagery must be registered.

The registered version of Imagery has many advantages, free technical support and free maintenance upgrades. Also several functions have extended functionality, and those functions marked as 'registered version only' are fully functional. The registered version is also less verbose and reports all messages to the calling application.

Upon registration you will receive the fully functional Imagery DLL. As mentioned you will receive any maintenance upgrades free. If you are interested in registering Imagery and would like more information on the registered version, e-mail **Support**

The registration fee of US $50.00 (or AUS $65.00) for a developers version or US $10.00 for the end user version may be paid by mail using the **Registration Form**.

Site and source code licensing is also available contact .**Support** for more information.

Registration Form

Mail this form to:

Ursus Computing Pty. Ltd.
PO BOX 116
Adelaide 5000
South Australia
Australia

Please Include a cheque/money order for US $50.00 (or AUS $65.00) for the development version, or US $10.00 (or AUS $15.00) for the end-user version. The Cheque should be made out to Ursus Computing Pty. Ltd.

Full Name: _____

Company Name: _____

Development Environment: _____

Development/Target Hardware configuartion:

_____
_____
_____
_____

E-Mail Address: _____

Full Postal Address:

_____
_____
_____
_____

How did you discover/get Imagery:

_____

Recommendations for future versions:

_____
_____
_____

Method of delivery: _____

Credits and Copyright

Imagery:

TIFF Library:

JPEG Library:

PNGLib:

Support

Postal Address

Registration Forms and payment of registration should be sent to:

Ursus Computing Pty. Ltd.
P.O. Box 116
Adelaide 5000
South Australia
Australia

E-Mail Address

Questions relating to Imagery should be sent to:

ursus@magna.com.au

$ Stuff to do# Stuff_to_doK Stuff to do