ImgDLL

This is a Win32 DLL which provides a variety of image file and image processing functions. The functions are described below.

This DLL has been used in VC++ 4 and 5, Visual BASIC, Borland, Watcom and Access.

What the package includes

- ImgDLL.DLL - the image library
- ImgDLL.H - the main header file
- ImgDLL.LIB - the library file, used for static linking of the DLL
- ImgDemo.ZIP - a small MFC sample application, written in VC++ 5.0, that demonstrates image file reading , processing and writing using ImgDLL.DLL.
- ImgDemo.EXE - the compiled ImgDemo application
- ImgDLLVx.RTF – this file.
- Function.TXT - brief descriptions of the functions and the versions in which they were introduced.
- Free tech support via e-mail
- Free bug fixes and upgrades on request
- See http://www.smalleranimals.com/imgboard/imgboard.html for the ImgDLL discussion board

Cost

What it costs for an unlimited license (one copy, use as you please) :

- $15, US

The DLL on my website is a full, working version. However, to use it properly, you will need an access key, which you will use when you initialize the DLL. The access key is what you get for your $15. Without the key, the ability to read and write images will be limited to images of less than 100 pixels in either dimension.

Source

Source is not included in the regular ImgDLL package. Source may be purchased separately from ImgDLL at a cost of $45 ($30 for registered users).

Static Lib Version

This library is also available as a set of static LIBs. The release and debug multi-threaded versions of two static libraries are included. All functionality is the same as in the DLL; all pricing and access-key policies are the same as for the DLL. Contact smallest@smalleranimals.com for info on how to obtain the static LIB version.

Product Information

- JPG info from IJG 6b. Full IJG source available at the simtel archives.
- PNG code from LibPng v1.0.0
- TIFF code from LibTIFF 3.4 beta 024
- PCX code from Bob Johnson
- Color quantizer based on Dennis Lee's DL1Quant.
- All other code Copyright 1997, 1998 Smaller Animals Software

---

RGB

When ImgDLL refers to something as being an “RGB” buffer, it means that the size of this buffer is exactly :

3 * imageWidth * imageHeight BYTEs.

If you are getting images which seems to be tripled, or skewed or only 1/3 wide or high, make sure you have accounted for the “3” when allocating space for your image.

Sample C/C++ code

See the ImgDemo project for a more thorough example.

Copy a BMP file to a JPG file

```
UINT width, height;

// read the BMP to a packed buffer of RGB bytes
HGLOBAL hpRGB=ImgDLLReadRGBFromBMP(inPath, &width, &height);

// test for error
if (hpRGB==NULL) {
       // error
} else {
    // lock the global memory
    BYTE * pRGB = (BYTE *)GlobalLock(hpRGB);
    if (pRGB==NULL) {
        // error! clean up
        GlobalFree(hpRGB);
    } else {

        // add an optional JPEG_COM text string
        ImgDLLAddJPGText(“ImgDLL Sample”);

        // save it
        BOOL ok = ImgDLLSaveRGBToJPG(outPath, pRGB,
                                     width, height,
```

```
                                        qualityVal,
                                        TRUE);

        // clear JPG output text so that we don't
        // write it to the next JPG file by mistake
        ImgDLLClearJPGOutputText();

        // test for error
        if (!ok)
             // error

        GlobalUnlock(hpRGB);
        GlobalFree(hpRGB);
    }
}
```

Read a PNG file and its text fields

```
// read the PNG to a packed buffer of RGB bytes
HGLOBAL hpRGB=ImgDLLReadRGBFromPNG(inPath, &width, &height);

// test for error
if (hpRGB==NULL) {
   // error
} else {
       // how many fields were in that file?
       UINT numText = ImgDLLGetPNGTextCount();

       CString text="";
       for (UINT t=0;t<numText;t++) {
              // get a key
              HGLOBAL hKey = ImgDLLGetPNGKey(t);
              if (hKey!=NULL) {
                     char * pKey = (char *)GlobalLock(hKey);
                     text+="Key:";
                     text+=pKey;
                     text+="\n";
                     GlobalUnlock(hKey);
                     GlobalFree(hKey);
              }

              // get some text
              HGLOBAL hTxt = ImgDLLGetPNGText(t);
              if (hTxt!=NULL) {
                     char * pTxt = (char *)GlobalLock(hTxt);
                     text+"Text";
                     text+=pTxt;
                     text+="\n";
                     GlobalUnlock(hTxt);
                     GlobalFree(hTxt);
              }
       }

       ImgDLLClearPNGText();      // clear buffers, to be nice

       // display the text in a msg box
       AfxMessageBox(text);

       GlobalFree(hpRGB);         // free image memory
```

```
}
```

A note about HGLOBAL, GlobalLock, GlobalUnlock, GlobalFree.

ImgDLL passes images back to the caller in buffers that are allocated with GlobalAlloc(GHND,..). This means that, in order to get a pointer to the images, you *must* use GlobalLock.

Example :

```
// step 1
// allocate 10000 BYTEs
HGLOBAL hMemory;

hMemory  = GlobalAlloc(GHND, 10000);

if (hMemory==NULL)
{
error!
}

// step 2
// get a pointer to the memory
BYTE * pMemory;

pMemory = (BYTE *)GlobalLock(hMemory);

if (pMemory==NULL)
{
error!
}

// step 3
// now you can use pMemory as a pointer to 10000 BYTEs of data.
….

// when you're done….

       // step 4
// release the lock . this doesn't delete the memory!! you can
// always GlobalLock it again.
GlobalUnlock(hMemory);

// step 5
// clean up
GlobalFree(hMemory);
```

ImgDLL performs step 1 for you whenever it returns an HGLOBAL. It is up to you to at least perform steps 4 and 5 to release the memory back to the OS.

Single-line de/compression

As of v3.6, ImgDLL supports single-line JPG and BMP de/compression. This allows you to read and write JPG and BMP files a single line at a time.

Sample:

```
// read pic1.jpg
// create the decompression object
// DJIS = Decompress Jpeg Info Struct

UINT w,h;
HGLOBAL hJIS = ImgDLLCreateDJIS("C:\\pic1.jpg", &w, &h);
if (hJIS)
{
	// now that we know how big the image will be,
	// allocate a place to put it

	BYTE *pRGB = new BYTE[w * h * 3];
	UINT row = 0;
	BOOL ok = TRUE;
	UINT res = IMGOK;

	while (ok && (res==IMGOK))
	{
		// move our pointer to the next image row
		BYTE * rowPtr = pRGB + (row * w * 3);

		// read the next line
		ok = ImgDLLGetNextDJISLine(hJIS, rowPtr);

		row++;
		res = ImgDLLGetLastError();
	}

	// IMGNOLINES is set when you reach the end of the file
	if (res!=IMGNOLINES)
	{
		// error ?
	}

	// clean up. This is important!!
	ImgDLLDestroyDJIS(hJIS);

	// destroy the image, because we're not using it
	// in this example
	delete [] pRGB;
}
```

Single line writing is similar :

1. Get a CJIS object from ImgDLLCreateCJIS
2. Feed one RGB pixel line at a time to ImgDLLWriteNextCJISLine
3. When finished, clean up with ImgDLLDestroyCJIS

Note: BMP files are stored upside-down. This means you have to read and write the lines in reverse order. This is part of the BMP format and not necessarily an ImgDLL limitation.

Disclaimer

Use at your own risk. I make no guarantees as to the suitability of this software for any particular use. I find that it works fine for what I need it to do, but I cannot anticipate all uses to which people might try to put this. Any bugs, if reported to me, will be fixed as soon as possible. I can not be held responsible for any bugs or the consequences of any bugs which were not reported to me.

PLEASE!!! Report any bugs you find. I take great pains to insure that this is quality software. I don't want bugs!

Technical Support

I have been told this has been used successfully on VC++ 4.x, 5.0, Borland 5.0,3.0 C++, Borland's C++ Builder, Visual BASIC, and MS Access.

I know VC++ 1.5, 4.x and 5.0, I don't know Visual BASIC or Borland's C++. From what I understand, though, this is a standard C-interfaced DLL and thus should be compatible with all environments - correct me if I'm wrong, I don't want to mislead anyone here.

I will try to help you as much as I can. Please do not ask the IJG or PNG people for help with this, nor should you ask anyone else who's code I have included - they have provided their code with no obligations to support what I do with it.

Error: Reference source not found

ImgDLL Functions as of V3.7

General DLL functions	
ImgDLLGetLastError	Find out what caused a failure
ImgDLLInitDLL	Initialize the DLL
ImgDLLStringInitDLL	Initialize with a string
ImgDLLSetCallback	Set a callback function

BMP file functions	
ImgDLLGetBMPDimensions	Find size of BMP image
ImgDLLReadRGBFromBMP	Read 24-bit from BMP
ImgDLLSaveColormappedToBMP	Write 1,4,8 bit BMP
ImgDLLSaveDIBColormappedToBMP	Save DIB to BMP
ImgDLLSaveRGBToBMP24	Write 24-bit BMP
ImgDLLReadHBITMAPFromBMP	Read HBITMAP from BMP
ImgDLLCreateDBIS	Create a single-line BMP read struct
ImgDLLGetNextDBISLine	Read a single scanline
ImgDLLDestroyDBIS	Clean up
ImgDLLCreateCBIS	Create a single-line BMP write struct
ImgDLLWriteNextCBISLine	Write a single scanline
ImgDLLDestroyCBIS	Clean up

JPG file functions	
ImgDLLGetJPGDimensions	Find size of JPG image
ImgDLLReadRGBFromJPG	Read JPG to 24-bit buffer
ImgDLLGrayscaleToJPG	Save 8-bit gray image to JPG
ImgDLLSaveRGBToJPG	Save 24-bit buffer to JPG
ImgDLLSetJPGErrorMsgBox	En/Disable the JPG error boxes
ImgDLLReadRGBFromJPGMem	Read JPG stream to RGB
ImgDLLSaveRGBToJPGMem	Create JPG stream in memory
ImgDLLSetJPGDCT	Control JPG compression
ImgDLLReadHBITMAPFromJPG	Read JPG to HBITMAP
ImgDLLRead8bitGrayscaleFromJPG	Read JPG to 8-bit grayscale
ImgDLLCreateDJIS	Create a single-line JPG read struct
ImgDLLGetNextDJISLine	Read a single scanline
ImgDLLDestroyDJIS	Clean up
ImgDLLCreateCJIS	Create a single-line JPG write struct
ImgDLLWriteNextCJISLine	Write a single scanline
ImgDLLDestroyCJIS	Clean up
ImgDLLAddJPGText	Add optional JPEG_COM marker text
ImgDLLGetJPGText	Return JPG text read from file
ImgDLLGetJPGInputTextCount	How many JPG text fields were read
ImgDLLClearJPGInputText	Clear input text strings
ImgDLLClearJPGOutputText	Clear JPG output text

TIFF file functions	
ImgDLLGetTIFFDimensions	Get TIFF image dimensions
ImgDLLTIFFToRGB	Read 24-bit from TIFF
ImgDLLSaveColormappedToTIFF	Write 8 bit TIFF
ImgDLLSaveRGB24ToTIFFRGB	Write 24 bit to TIFF

PNG file functions	
ImgDLLGetPNGDimensions	Get PNG image dimensions
ImgDLLReadRGBFromPNG	Read a PNG file to an RGB buffer
ImgDLLSave8BitToPNG8Bit	Save 8-bit image to PNG 8-bit
ImgDLLSaveRGB24ToPNGRGB	Save RGB 24-bit to PNG RGB 24-bit
ImgDLLSaveToPNG	Save image buffer to PNG file
ImgDLLPNGSetScreenGamma	Set the screen gamma for PNG reads
ImgDLLPNGSetDefBackground	Set default background color for PNG reads
ImgDLLGetPNGTextCount	Find out how many text fields were read during the last PNG file read operation.
ImgDLLGetPNGKey	Fetch text field keys from the ImgDLL PNG input text buffer.
ImgDLLGetPNGText	Fetch text fields from the ImgDLL PNG text input buffer.
ImgDLLClearPNGText	Remove all PNG text fields from the input and output ImgDLL PNG text buffers.
ImgDLLAddPNGText	Add a new text field to the ImgDLL PNG output text buffer.

PCX file functions	
ImgDLLGetPCXDimensions	Get the width and height of a PCX file
ImgDLLReadRGBFromPCX	Read a PCX file to a 24-bit buffer
ImgDLLSaveRGB24ToPCXRGB	Save RGB-24 image to a 24-bit PCX file
ImgDLLSaveColormappedToPCX	Save 8-bit colormapped to PCX -bit

Buffer manipulation functions	
ImgDLLDIBToRGB	Convert a DIB to RGB
ImgDLLRGBToDIB	Convert a 24-bit RGB buffer to a DIB
ImgDLLDWORDAlignBuf	DWORD align a buffer of pixels
ImgDLLDWORDAlignBufBytes	DWORD align a buffer of bytes
ImgDLLRGBFromDWORDAligned	un-DWORD align a buffer

Image processing functions

ImgDLLMakeGrayScale	Create 24-bit grayscale image
ImgDLLMake8BitGrayScale	Create 8-bit gray from 24-bit RGB
ImgDLLRGBFrom8Bit	Apply palette to 8-bit image
ImgDLLQuantizeRGBTo8Bit	Generate 8-bit image from 24-bit
ImgDLLCountRGBColors	Count colors in RGB image
ImgDLLColorSubRGB	Replace one color with another
ImgDLLVerticalFlipBuf	Vertically flip a buffer
ImgDLLHorizontalFlipRGB	Horizontal flip an RGB image
ImgDLLBlurRGB	Blur an RGB image
ImgDLLResizeRGB	Resize an RGB image
ImgDLLResizeRGB2	Resize an RGB image
ImgDLLSharpenRGB	Sharpen an RGB image
ImgDLLHistogramEqualizeRGB	Apply EQ to an RGB image
ImgDLLRotateRGB	Rotate an RGB image
ImgDLLQuickRotateRGB	Rotate an RGB image 90, 180 or 270 deg.
ImgDLLCropRGB	Crop an RGB image
ImgDLLZoomRGB	Zoom in on an RGB image
ImgDLLApplyConvolutionFilter	Apply an arbitrary filter to an image
ImgDLLApplyMatrixToRGB	Apply 3x3 matrix to an RGB image
ImgDLLApplyLUTToRGB	Apply 256-entry LUT to an RGB image
ImgDLLOverlayRGB	Overlay one RGB image on another
ImgDLLOverlayRGBTransparent	Overlay with transparency
ImgDLLDrawTextOnRGB	Render text onto RGB image
ImgDLLDrawTextOnRGB2	Render text using a valid LOGFONT *
ImgDLLSetAlphaChannelToImage	Fill Alpha channel of an RGBA image with 8-bit image
ImgDLLRGBAFromRGB24	Create RGBA from an RGB image
ImgDLLBGRFromRGB	Swap red and blue in RGB
ImgDLLGet8BitPalette	Generate 8-bit palette from an RGB image
ImgDLLGetBrightnessHistogram	Get the brightness histogram for an image
ImgDLLGetChannelHistogram	Get the histogram for a single channel
ImgDLLDecimateRGB	Reduce an RGB image, using a block averaging algorithm
ImgDLLDecimateRGB2	Reduce an RGB image, using a block averaging algorithm

HBITMAP functions	
ImgDLLLoadResourceBMP	Load resource BMP with palette
ImgDLLHBITMAPToGGB	Convert HBITMAP to RGB
ImgDLLRGBToHBITMAP	Convert RGB to HBITMAP
ImgDLLDCToRGB	Grab a section of a DC to an RGB buffer

Image output functions	
ImgDLLDrawHBITMAP	Draw HBITMAP to screen
ImgDLLDrawRGB	Draw RGB to screen
ImgDLLDrawRGB2	Fast RGB draw
ImgDLLDrawTranparentRGB	Draw RGB with one color transparent
ImgDLLDrawTransparentHBITMAP	Draw HBITMAP with one color transparent

GIF Notes

The Unisys corporation charges $750 in advance royalties for a GIF/LZW software license for “toolkit” software. ImgDLL would qualify as toolkit software. This is cheap, compared to the $2500 in advance royalties Unisys charges for end-user software. That is, if you want to use GIF in your own software, you would have to pay Unisys $2500 up-front, as an advance on the per-copy royalties they will collect on your software. (this is as of 4/98).

PNG notes

Alpha channels:

PNG permits images to be stored with an alpha channel. This is an extra BYTE (or 2 BYTEs, depending on the bit depth of the image) that controls the blending of the image with the background. Depending on the reader, this could be a single solid color or an actual image. A web-browser might be able to display a semi-transparent PNG image over the background image of a web-page. An alpha value of 255 means fully opaque, a value of 0 means fully transparent (no image, all background).

Gamma:

Gamma is a complicated topic. In **extremely** simple terms, it is a brightness response value of a monitor or other output device. Every output device has a certain Gamma value . PC monitors tend to have values near 2.2, Mac monitors tend to have values of around 1.8, SGI - 1.5, Next 1.0, etc.. This means that a given GIF image (for example) will be drawn differently on each monitor. This can be a problem, for some images and applications. PNG allows the creator of an image to specify the gamma value of the device it was created on. All other (competent) PNG readers should then be able to adjust the output image so that it matches the initial on-screen appearance. Some PNG readers are not fully gamma capable. There are various methods to find the gamma value for your own particular monitor. Most of them are inaccurate. Paint Shop Pro has a gamma value section in its on-line Help.

NOTE : when specifying a file gamma value, don't use the actual gamma value for your monitor, use the inverse (1 / screen_gamma) ! File gammas must be < 1.0.

Color Type:

PNG allows images to be saved in a number of different formats. The benefits of each format depend on the image you are saving.

Using ImgLibSaveToPNG,

if you want to save an ordinary RGB 24-bit image, specify :

PNG_COLOR_TYPE_RGB	for color type
8	for bit depth
inWidthPix * 3	for width bytes
NULL	for palette

if you are saving a 32-bit RGBA image, specify :

PNG_COLOR_TYPE_RGB_ALPHA	for color type
8	for bit depth
inWidthPix * 4	for width bytes
NULL	for palette

if you are saving an 8-bit image with a 256 color palette, specify :

PNG_COLOR_MASK_PALETTE	for color type
8	for bit depth
inWidthPix	for width bytes
a pointer to your palette	for palette

Background:

The background color to be used in this image is set with ImgDLLPNGSetDefBackground. The meaning of the value you set is dependent on the colorType value you use for your image data.

Interlacing:

PNG defines two kids of interlacing - none, and Adam7. These have no effect on the actual image that is saved: they are used by PNG readers to control how the image is displayed as it is read. Adam7 is a 2-D interlacing that allows images to be drawn in 2D (not simply vertically, like JPG or GIF) as they are read.

Text fields:

PNG allows a file to have an arbitrary number of text fields. These fields consist of a 1-79 character "key" and a text buffer of unlimited size. The text can be compressed or uncompressed. Keys should be plain ASCII, no control or non-printable chars. You may add as many fields as you wish.

The keywords that are given in the PNG Specification are:

Title	Short (one line) title or caption for image
Author	Name of image's creator
Description	Description of image (possibly long)
Copyright	Copyright notice
Creation Time	Time of original image creation
Software	Software used to create the image
Disclaimer	Legal disclaimer
Warning	Warning of nature of content
Source	Device used to create the image
Comment	Miscellaneous comment

ImgDLL maintains two global PNG text field buffers. One is used to hold text fields that will be written with the next PNG file write operation; these are set with ImgDLLAddPNGText . The other buffer contains text fields that were read during the last file read operation; these are accessed with ImgDLLGetPNGKey and ImgDLLGetPNGText. You can only read the strings in the input buffer, and you can only add strings to the output buffer.

To write a PNG file with text fields, you need to do two things prior to the PNG file write call. First, call ImgDLLClearPNGText(). This clears the ImgDLL global PNG text buffers. Second, call ImgDLLAddPNGText(), once for each text field you want to be written.

The sample C/C++ code shows how to read the text fields in a PNG file.

Bottom line:

PNG is a format with a great deal of flexibility. As with most flexible formats, it requires that you know what you're doing before you can truly take advantage of it.