Image tools library is easy to use graphics library. It comprises routines for reading, writing, converting, implementing image effects and compression. The library is written in C language and as such is intended to be used with C compilers compatible with Microsoft C and Borland C. There is also Windows Dynamic Link Library (DLL) which can be used with any language that can use DLLs. Image tools library is specifically designed for Windows programmers and it is intended for use in Windows 3.1.

INSTALLATION AND SETUP

Installation of Image Tools library is very simple. You should just copy appropriate ZIP file representing the library which you want to use in desired directory. After switching yourself into Image Tools directory you should invoke PKUNZIP program to decompress it. You must decompress the library using -d switch to create appropriate sub directory (for example: PKUNZIP -D IMAGE_BC to decompress Borland C library). After decompression you will have library and appropriate include files in that directory as well as sample sub directory containing ITDEMO program with sample source code.

All libraries are compiled in LARGE memory model so your application should also use it.

SYSTEM REQUIREMENTS

Image Tools doesn't have any special system requirements beside those already set by the compiler of your choice. However, one thing must be noted. The maximum size of the single block of memory is 16MB therefore the size of the image which you will want to manipulate can not exceed this limit. In order to efficiently use Image Tools you should check that application built with it has enough memory resources to perform it's task. It doesn't matter weather this memory is virtual (on disk) or RAM.

QUICK START

Image Tools is designed with easy-to-use in mind. For example to read any of the supported file formats you must know how to use just one routine. It is ImageLoad routine which takes 3 parameters LPFILEINFO lpFileInfo, LPFRAME lpFrame, short nFrame. This routine is all it takes to read TIF, TGA, PCX, GIF, BMP, FLI or FLC images.
What should one do to read in image in TGA (Truevision Targa) format and write it out in 8-bit BMP format for example. At the beginning of your Windows C or C++ program you should put appropriate include statements :

```
    #include "IMAGE_LS.H"      // For Load & Save library
    #include "IMAGE_EF.H"      // For Effects library
```

Now you must declare FILEINFO structure:

```
        FILEINFO FileInfo;
```

You must also declare FRAME structure:

```
        FRAME Frame;
```

Must also fill FILEINFO structure with the name of the file containing TGA image and initialize LPFILEINFO pointer to it:

```
        strcpy( FileInfo.szFileName, "MYPIC.TGA");
        FileInfo.wFileType = FILE_TGA;
```

Now you are ready to read the image by calling ImageLoad:

```
        ErrCode=ImageLoad(&FIleInfo, &Frame, 1);
```

Unless ImageLoad has returned error code (ErrCode != 0), you will have image loaded into memory. Pointer to the exact memory location is in the FRAME structure. You can access it with Frame.hPicture which is a global handle of a memory block where the 24-bit image is stored.

Now you should fill FILEINFO structure with the name of the output BMP file:

```
        strcpy( FileInfo.szFileName, "MYPIC.BMP");
        FileInfo.wFileType = FILE_BMP;
```

And call ImageSave routine:

```
        ErrCode=ImageSave(&FileInfo, &Frame);
```

This is it.

LOADING AND SAVING IMAGE FILES

As you could see in quick start implementing routines for loading and saving image files is pretty straightforward. Here we will explain in detail various data structures and parameters used to invoke these routines.

Image Tools routines read the complete image in a block of memory where it can operate on it. This puts certain requirements to your computer equipment, Windows operating system and your program. To manipulate large images you must have large chunks of memory at your disposal. This required memory can be free RAM or virtual memory on your disk. To use virtual memory you must specify it through Windows Control Panel (386 Enhanced program). As Windows limits maximum amount of a single chunk of memory to 16MB that restricts your image size also to 16MB. To access this memory ImageTools uses huge pointers. If your application is going to access images directly in memory it will have to use huge pointers also.

There are just two structures you must be familiar with to be able to use Image Tools library.

FRAME STRUCTURE

Routines for loading images automatically fill all the data in this structure. User must only declare this structure.

```
typedef struct {
   WORD          iWidth,
                 iHeight;
/* Image dimensions. Width and height of an image. This data is filled by LoadImage routine. Various effects routines can change these values also. */
   HANDLE        hPicture;
/* Handle for a chunk of memory allocated by image load routines. Using Windows GlobalLock routine on this handle returns pointer to a block of memory where the actual image resides */
   HANDLE        hBackground;
/* Handle for a chunk of memory allocated by image load routines. Using Windows GlobalLock routine on this handle returns pointer to a block of memory where the actual material channel data resides.  Each pixel has a corresponding byte in this block of memory which represents it's alpha value (amount of transparency). This data exists only in TGA images and is useful only if you want to implement transition effects. */
   WORD          iNumBits,     /* 8 or 24 color */
                 nPalEntries;  /* number of optimized palette entries */
   HPALETTE      hPalette;     /* handle for windows logic palette */
} FRAME, FAR *LPFRAME;
```

FILEINFO STRUCTURE

This structure is needed only for routines that do loading and saving of image files.

```
typedef struct {
          char szFileName[MAX_FILE_NAME+1];
/* C string containing the name of the file. This field must be filled by user before calling load and save routines. */
          WORD wFileType;
/*  Holds file type inf. These defines are in image_ls.h file
          #define  BMP_FILE                 1
          #define  GIF_FILE                 2
          #define  PCX_FILE                 3
          #define  TGA_FILE                 4
          #define  TIF_FILE                 5
          #define  FLI_FILE                 6
          #define  FLC_FILE                 7
          #define  JPG_FILE                 8
*/
        WORD wCompression;
/* Defines weather image data will be compressed or not when ImageSave routine is called. For JPEG compression it
represents image quality in the range 0-100 */
        WORD nFrames;
/* Parametar used for FLC/FLI file types only. This is a number of animation frames in the animation file. */
} FILEINFO, FAR *LPFILEINFO;
```

ImageLoad routine

ERR_RETURN ImageLoad(LPFILEINFO *lpFileInfo*, LPFRAME *lpFrame*, short *nFrame*);
lpFileinfo pointer to FILEINFO structure
lpFrame pointer to FRAME structure
nFrame frame number in FLI or FLC file

With lpFileinfo pointer to FILEINFO structure and lpFrame pointer to FRAME structure you are already familiar. You must only fill in szFileName C string within FILEINFO structure pointed to by lpFileinfo with the name of the image file. Depending on the wFileType member of FILEINFO structure ImageLoad will invoke appropriate routine fro loading desired file type. For example, if the wFileType=1 then routine for reading BMP files will be invoked. If you want to load certain frame from FLI or FLC file you should supply nFrame parameter representing the frame number which you want to load from specified FLI or FLC file. Otherwise this parameter should be 1.
ImageLoad routine always returns 24-bit image in FRAME structure regardless of type of image on the disk (It converts picture with different number of bit planes to 24 bit).
Returns 0 on success or error number on failure. For error number description look at liberror.hh file.

ImageSave

ERR_RETURN S_TYPE ImageSave(LPFILEINFO *lpFileInfo*, LPFRAME *lpFrame*);
lpFileinfo pointer to FILEINFO structure
lpFrame pointer to FRAME structure

With lpFileinfo pointer to FILEINFO structure and lpFrame pointer to FRAME structure you are already familiar. You must fill in szFileName string within FILEINFO structure pointed to by lpFileinfo with the name of the image file. Depending on the wFileType member of FILEINFO structure ImageSave will invoke appropriate routine for saving desired file type. For example, if the wFileType=1 then routine for saving BMP files will be invoked. User must also specify weather compression will be used when storing image data. Setting wCompression member of the FILEINFO structure to 0 defines no compression and 1 to go on with compression. With JPEG format this parameter defines quality and must be in range from 0 to 100.
If you want to save image in 8-bit format you must first convert it to 256 colors using routine MapTo8Bits.
Returns 0 on success or error number on failure. For error number description look at liberror.hh file.

Here is the example:

```
                                FRAME Frame24, Frame8;
```

```
                                        .
                                        .
                                        .

                                        ImageLoad(&Frame24, &FileInfo, 1);
                                        if(MapTo8Bits(&Frame24, &Frame8))
                                                goto ERROR;
                                        }
                                        .
                                        .
                                        .
                                        ImageSave(&Frame8, &FileInfo);
                                        .
                                        .
                                        .
ERROR:
                                        if(Frame.hNewPic) GlobalFree(Frame.hNewPic);
                                        MessageBox(hWndMain,"Could not convert image to 8 bit","ERROR
OCCURED",MB_OK);
```

FLXGetInfo

ERR_RETURN S_TYPE FLXGetInfo(LPFILEINFO *lpFileInfo*, LPFRAME *lpFrame*);
lpFileinfo pointer to FILEINFO structure
lpFrame pointer to FRAME structure

This routine fills members of FRAME and FILEINFO structures with data referring to desired FLI/FLC file. You must fill in szFileName string within FILEINFO structure pointed to by lpFileinfo with the name of the FLI or FLC file. wFileType member of FILEINFO structure must be set either to FLI_FILE (6) or FLC_FILE (7).
Returns 0 on success or error number on failure. For error number description look at liberror.hh file.

FLXLoad

ERR_RETURN S_TYPE FLXLoad(LPFILEINFO *lpFileInfo*, LPFRAME FAR* *lplpFrame*, short *nStartFrame*,short *nEndFrame*);
lpFileinfo pointer to FILEINFO structure
lplpFrame pointer to array of LPFRAME pointers
nStartFrame starting frame number
nEndFrame ending frame number

This routine loads images from FLI or FLC files and fills members of LPFRAME and LPFILEINFO structures with data regarding dimensions and number of images stored in FLI or FLC file.You must fill in szFileName C string within FILEINFO structure pointed to by lpFileinfo with the name of the FLI or FLC file. wFileType member of FILEINFO structure must be set either to FLI_FILE (6) or FLC_FILE (7). nStartFrame and nEndframe are numbers of the first and the last frame which will be loaded. All frames between and including nStartFrame and nEndFrame will be loaded. You must take care that lplpFrame points to an array with enough LPFRAME pointers to FRAME structures to hold data for every frame which will be loaded. Number of LPFRAME elements in array pointed to by lplpFrame must be equal to the number of frames in a range defined by nStartFrame and nEndFrame. The simplest way to do this is following example:

```
	FRAME aFrames[10];
	LPFRAME lpFrames[10];
	for(i=0;i<10;i++) lpFrames[i]=&aFrames[i];
	FlxLoad(&FileInfo, lpFrames, 10, 19);
```
When this routine finishes, all frames contain 24-bit pictures.
Returns 0 on success or error number on failure. For error number description look at liberror.hh file.

FLXCreate

ERR_RETURN FLXCreate(LPFILEINFO *lpFileInfo*, LPFRAME *lpFrame*);
lpFileinfo pointer to FILEINFO structure
l*pFrame* pointer to FRAME structure

This routine creates FLI or FLC file and fills it with one frame. You must fill in szFileName string within FILEINFO structure pointed to by lpFileinfo with the name of the FLI or FLC file. wFileType member of FILEINFO structure must be set either to FLI_FILE (6) or FLC_FILE (7). This routine does not put the ring frame on the end, and it's used only to create file for appending.
Returns 0 on success or error number on failure. For error number description look at liberror.hh file.

FLXSave

ERR_RETURN S_TYPE FLXSave(LPFILEINFO lpFileInfo, LPFRAME FAR* lplpFrame, short nFrames);
lpFileinfo pointer to FILEINFO structure
lplpFrame pointer to array of LPFRAME pointers
nFrames number of frames which will be saved

This routine saves images to FLI or FLC files. It calls FLXCreate routine for the first frame and FLXAppend for next frames. You must fill in szFileName string within FILEINFO structure pointed to by lpFileinfo with the name of the FLI or FLC file. wFileType member of FILEINFO structure must be set either to FLI_FILE (6) or FLC_FILE (7). nFrames defines number of frames which will be saved to FLI or FLC file. Number of LPFRAME elements in array pointed to by lplpFrame must be equal to the nFrames parameter.
Returns 0 on success or error number on failure. For error number description look at liberror.hh file.

FLXAppend

ERR_RETURN S_TYPE FLXAppend(LPFILEINFO lpFileInfo, LPFRAME FAR* lplpFrame, short nFrames);
lpFileinfo pointer to FILEINFO structure
lplpFrame pointer to array of LPFRAME pointers
nFrames number of frames which will be saved

This routine append images to end of the FLI or FLC files.You must fill in szFileName string within FILEINFO structure pointed to by lpFileinfo with the name of the FLI or FLC file. wFileType member of FILEINFO structure must be set either to FLI_FILE (6) or FLC_FILE (7). nFrames defines number of frames which will be saved to FLI or FLC file. Number of LPFRAME elements in array pointed to by lplpFrame must be equal to the nFrames parameter.

Returns 0 on success or error number on failure. For error number description look at liberror.hh file.

Note: Support for JPEG compression/decompression and JPG file format:
this software is based in part on the work of the Independent JPEG Group

EFFECTS

To implement effects you must be familiar with FRAME structure. (For description of this structure look at the previous chapter.) You can perform following effects:

Effect	**Routine name**
Blure effect	EffBlure
Brightness	EffBrightness
Cotrast	EffContrast
Crop	EffCrop
Edge detection	EffEdgeDetection
Flip	EffFlip
Move	EffMove
Rotate	EffRotate
Scale	EffScale
Zoom	EffZoom
Sharpen	EffSharpen
Convert 24 to 8 bit color	MapColorsTo8Bits
Build 8 bit palette	BuildPalette

In most effects routines you must supply 2 LPFRAME pointers. One points to the FRAME containing information on source frame and the other contains information (including pointer to a block of memory which will hold the image) on the frame which will hold resulting image. For the resulting image user must fill in contents of the FRAME structure. It means that the user must fill in iWidth and iHeight members with the same values which are in the FRAME structure representing source image. User must also allocate memory which will hold the resulting image. It can be done using AllocFrame routine. AllocFrame takes pointer to FRAME structure as the only parameter. User must fill only iWidth and iHeight members of the FRAME structure.
All effects routines return 0 on successful completion.

Allocating memory for images

SHORT S_TYPE AllocFrame(LPVOID lpFrame);
LPFRAME lpFrame; pointer to FRAME structure with set iWidth and iHeight members

User must fill only iWidth and iHeight members of the FRAME structure pointed to by lpFrame. AllocFrame will calculate amount of needed memory to hold the image and set hPicture member of the FRAME structure to be equal to the memory handle for that chunk of memory.

Blure Effect

short S_TYPE EffBlure(LPVOID lpSF, LPVOID lpRF, WORD wBlureValue)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame
WORD wBlureValue;	intensity of blure effect (number of surrounding pixels affected)

User must fill in iWidth and iHeght parameters in the FRAME structure pointed to by lpRF. User must

also allocate enough memory which will hold resulting image. It can be done by AllocFrame routine. Handle for that block of memory must be stored in the hPicture member of the FRAME structure pointed to by lpRF.
wBlureValue specifies the amount of blur. Actually it is number of surrounding pixels which will be affected.

Brightness Effect

short S_TYPE EffBrightness(LPVOID lpSF, LPVOID lpRF, short sAmount)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame
short sAmount;	ammount to brighten image

User must fill in iWidth and iHeght parameters in the FRAME structure pointed to by lpRF. User must also allocate enough memory which will hold resulting image. It can be done by AllocFrame routine. Handle for that block of memory must be stored in the hPicture member of the FRAME structure pointed to by lpRF.
sAmount parameter represents integer number which will be added to each R, G and B component. It should be in the range -255, +255.

Contrast Effect

short S_TYPE EffContrast(LPVOID lpSF, LPVOID lpRF, short wContrast)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame
short wContrast;	contrast factor in %

User must fill in iWidth and iHeght parameters in the FRAME structure pointed to by lpRF. User must also allocate enough memory which will hold resulting image. It can be done by AllocFrame routine. Handle for that block of memory must be stored in the hPicture member of the FRAME structure pointed to by lpRF.
wContrast parameter is percentage of the contrast which should be applied.

Crop Effect

short S_TYPE EffCrop(LPVOID lpFrame1, LPVOID lpFrame2, WORD wX1, WORD wY1,
WORD wX2, WORD wY2)

LPFRAME lpfrFrame1;	long pointer to original image frame
LPFRAME lpfrResultFrame;	long pointer to result image frame
WORD X1, X2, Y1, Y2	crop window coordinates

Edge detection effect

short S_TYPE EffEdgeDetection(LPVOID lpSF, LPVOID lpRF, WORD iTreshold, DWORD colColor1, DWORD colColor2)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame
WORD wFactor;	Treshold factor [0,100]
COLORREF colColor1;	foreground color
COLORREF colColor2;	background color

User must fill in iWidth and iHeght parameters in the FRAME structure pointed to by lpRF. User must also allocate enough memory which will hold resulting image. It can be done by AllocFrame routine. Handle for that block of memory must be stored in the hPicture member of the FRAME structure pointed to by lpRF.
wFactor represents treshold factor for detecting edges in the source image. More higher the wFactor more edges will be detected.
colColor1 represents the RGB color which will be used to represent the edges in the resulting image.
colColor2 represents the RGB color which will be used as the background in the resulting image.

Flip effect

short S_TYPE EffFlip(LPVOID lpSF, LPVOID lpRF, WORD fVertical, WORD Horizontal)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame
WORD fVertical;	flip vertical? [1=flip, 0=don't flip]
WORD fHorizontal;	flip horizontal? [1=flip, 0=don't flip]

User must fill in iWidth and iHeght parameters in the FRAME structure pointed to by lpRF. User must also allocate enough memory which will hold resulting image. It can be done by AllocFrame routine. Handle for that block of memory must be stored in the hPicture member of the FRAME structure pointed to by lpRF.
If fVertical is 1 then vertical flip will be done.
If fHorizontal is 1 then horizontal flip will be done.

Move effect

short S_TYPE EffMove(LPVOID lpSF, LPVOID lpRF, short sToX, short sToY)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame
short sToX, sToY;	move amount in pixels

User must fill in iWidth and iHeght parameters in the FRAME structure pointed to by lpRF. User must also allocate enough memory which will hold resulting image. It can be done by AllocFrame routine. Handle for that block of memory must be stored in the hPicture member of the FRAME structure pointed to by lpRF.
Image will be moved for sToX (X coordinate) pixels and sToY (Y coordinate) pixels in x and y directions. New origin of the image will be at sToX and sToY coordinates.

Rotate effect

short S_TYPE EffRotate(LPVOID lpSF, LPVOID lpRF, short sCenterX, short sCenterY, short wEndAngle)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame
short sCenterX, sCenterY;	center of rotation
short wEndAngle;	rotation angle in degrees [-180,180]

User must fill in iWidth and iHeght parameters in the FRAME structure pointed to by lpRF. User must also allocate enough memory which will hold resulting image. It can be done by AllocFrame routine. Handle for that block of memory must be stored in the hPicture member of the FRAME structure pointed to by lpRF.
sCenterX and sCenterY represent center of the rotation around which the image will be rotated.
wEndAngle is the degree of the rotation. It should be in the range -180, +180.

Scale Effect

short EffScale(LPVOID lpSF, LPVOID lpRF)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame

According to the iWidth and iHeight members of FRAME structures pointed to by lpSF and lpRF image is resized. New dimensions of the image will be lpRF->iWidth and lpRF->iHeight. Take care that there is sufficient block of memory allocated to hold scaled image.

Sharpen Effect

short S_TYPE EffSharpen(LPVOID lpSF, LPVOID lpRF, short wSharpen)

LPFRAME lpSF;	long pointer to original image frame
LPFRAME lpRF;	long pointer to result image frame
short wSharpen;	sharpen factor in %

User must fill in iWidth and iHeght parameters in the FRAME structure pointed to by lpRF. User must also allocate enough memory which will hold resulting image. It can be done by AllocFrame routine. Handle for that block of memory must be stored in the hPicture member of the FRAME structure pointed to by lpRF.
wSharpen is the percentage of increase in digital sharpness.

Zoom Effect

short EffZoom(LPVOID *lpSF*, LPVOID *lpRF* WORD *wX1*, WORD *wY1*, WORD *wX2*, WORD *wY2*)

LPFRAME *lpSF*; long pointer to original image frame

LPFRAME *lpRF*; long pointer to result image frame
WORD *X1, X2, Y1, Y2;* zoom window coordinates

Zoom window defined by coordinates X1, X2, Y1, Y2 is resized to fit the whole FRAME. After resizing X2-X1 will equal iWidth and Y2-Y1 will equal iHeight. Dimensions of original and zoomed image are the same. Pixels outside zoom window are lost in the resulting image.