NAME

imintro - general information on the SDSC image library

SYNOPSIS

#include "im.h"

cc *flags files* **-lsdsc -lim**

DESCRIPTION

The SDSC Image Library *libim.a* is a collection of portable C-language image manipulation tools. These tools allow image files in a variety of formats to be read in and written out and images to be manipulated while in memory. Manipulations include rotating, scaling, filtering, converting to/from RGB and gray-scale, cutting out pieces of images, changing color lookup tables, rolling color lookup tables, and so on.

The SDSC Image Library is made up of three sets of routines:

ImVfb	Manipulates Virtual Frame Buffers (images) while in memory
ImClt	Manipulates Color Lookup Tables while in memory
ImFile	Reads and writes image files and streams

Each of these is discussed in sections that follow.

VIRTUAL FRAME BUFFERS

Many visualization tools operate on frame buffers either to convert them from one metafile representation to another or to manipulate images, such as by clipping, filtering, or transformation. Each tool program requires some internal data structure. It is desirable for all such programs to use the same internal structure to allow sharing of techniques and code between applications. This shared data structure is called a **virtual frame buffer**.

Virtual Frame Buffer Design Goals

- The data structure must be able to store all necessary pieces of frame buffer information. This includes red, green, blue, color index, z-value, alpha-value, scalar data, as well as others.
- The data structure must be extensible to be able to hold more information when new situations arise.
- Programs must support multiple virtual frame buffers.
- The data structure should be as compact as possible, avoiding paging through a virtual frame buffer as much as possible.
- The data structure should only be visible inside of an application. Because we may want to change the internal format in the future, its structure should be hidden from the application with macros and procedure calls. It makes more sense for an application to query a red value from a pixel location in a virtual frame buffer with

  ```
  r = ImVfbQRed(v,p);
  ```

  instead of

  ```
  r = *( (unsigned char *) (p + v->v_roff ) );
  ```

  In the former case, a change to the definition of **ImVfbQRed** would require only a recompilation of the tools that use the redefinition.

- For efficiency, pointer arithmetic should be used to address pixels. Thus, constructs like the following should be possible when setting pixel color values:

```
ImVfb *v;
ImVfbPtr p, p1, p2;
int x1, y1, x2, y2;
 ...
p1 = ImVfbQPtr( v, x1, y1 );
p2 = ImVfbQPtr( v, x2, y2 );
for( p = p1; p <= p2; ImVfbSInc(v,p) )
{
     ImVfbSRed( v, p, 64 );
     ImVfbSGreen( v, p, 128 );
     ImVfbSBlue( v, p, 255 );
}
```

Allocating a Virtual Frame Buffer

A virtual frame buffer is allocated with a call to

```
ImVfb *v;
 ...
v = ImVfbAlloc( width, height, fields );
```

where *fields* defines what information each pixel will hold. This value is specified by or'ing together one or more of the following:

Constant	Meaning
IMVFBRGB	Stores red, green, blue values (0-255)
IMVFBALPHA	Stores an alpha value (0-255)
IMVFBWPROT	Stores a write protection (0 or non-zero)
IMVFBINDEX8	Stores a color index (0-255)
IMVFBINDEX16	Stores a color index (0-65534)
IMVFBMONO	Stores a monochrome (on/off) value
IMVFBZ	Stores a z-value (full integer)
IMVFBGRAY	Stores a gray scale value (0-255)
IMVFBGREY	Stores a gray scale value (0-255)
IMVFBIDATA	Stores an integer data value
IMVFBFDATA	Stores a floating-point data value

ImVfbAlloc allocates enough memory to hold such a frame buffer and sets up some internal information.

Information about a particular virtual frame buffer can be **S**et with

Call	Meaning
ImVfbSClt(v,c)	Attach (set) a pointer to a color lookup table

Information about a particular virtual frame buffer can be **Q**ueried with

Call	Meaning
ImVfbQWidth(v)	Returns the number of columns
ImVfbQHeight(v)	Returns the number of rows
ImVfbQFields(v)	Returns the fields mask
ImVfbQNBytes(v)	Returns the number of bytes per pixel
ImVfbQClt(v)	Returns a pointer to the color lookup table

Freeing a Virtual Frame Buffer

A virtual frame buffer's memory can be freed (deallocated) with a call to

```
ImVfb *v;
 ...
ImVfbFree( v );
```

Per-Pixel Storage

The per-pixel storage is a packed array of values. The storage convention assumes that the top row is row #0 and the left column is column #0. Pixels are stored like C-language 2D arrays: left-to-right across the row. Various values can be **S**et into a particular pixel (pointed to by a pixel pointer) within a particular virtual frame buffer by

Call	Meaning
ImVfbSRed(v,p,r)	Red (byte)
ImVfbSGreen(v,p,g)	Green (byte)
ImVfbSBlue(v,p,b)	Blue (byte)
ImVfbSAlpha(v,p,a)	Alpha-value (byte)
ImVfbSIndex8(v,p,i8)	Color index (byte)
ImVfbSIndex16(v,p,i32)	Color index (integer)
ImVfbSMono(v,p,m)	Monochromatic value (zero or one)
ImVfbSIndex(v,p,i)	ImVfbSIndex8 or ImVfbSIndex16
ImVfbSZ(v,p,z)	Z-value (integer)
ImVfbSGray(v,p,g)	Gray scale (byte)
ImVfbSGrey(v,p,g)	Gray scale (byte)
ImVfbSFData(v,p,f)	Floating-point data value
ImVfbSIData(v,p,i)	Integer data value

Various pixel values can be **Q**ueried within a particular virtual frame buffer by

Call	Meaning
ImVfbQRed(v,p)	Red (byte)
ImVfbQGreen(v,p)	Green (byte)
ImVfbQBlue(v,p)	Blue (byte)
ImVfbQAlpha(v,p)	Alpha-value (byte)
ImVfbQIndex8(v,p)	Color index (byte)
ImVfbQIndex16(v,p)	Color index (integer)
ImVfbQMono(v,p,m)	Monochromatic value (zero or one)
ImVfbQIndex(v,p)	ImVfbQIndex8 or ImVfbQIndex16
ImVfbQZ(v,p)	Z-value (integer)

ImVfbQGray(v,p)	Gray scale (byte)
ImVfbQGrey(v,p)	Gray scale (byte)
ImVfbQFData(v,p)	Floating-point data value
ImVfbQIData(v,p)	Integer data value

Pixel pointers can be **Q**ueried and **S**et by

Call	Meaning
ImVfbQPtr(v,x,y)	Returns a pointer to a particular pixel
ImVfbQFirst(v)	Returns a pointer to the first (UL) pixel
ImVfbQLast(v)	Returns a pointer to the last (LR) pixel
ImVfbQLeft(v,p)	Returns a pointer to a pixel one column left
ImVfbQRight(v,p)	Returns a pointer to a pixel one column right
ImVfbQUp(v,p)	Returns a pointer to a pixel one row up
ImVfbQDown(v,p)	Returns a pointer to a pixel one row down
ImVfbQNext(v,p)	Same as ImVfbQRight(v,p)
ImVfbQPrev(v,p)	Same as ImVfbQLeft(v,p)
ImVfbSInc(v,p)	Same as p = ImVfbQRight(v,p)
ImVfbSDec(v,p)	Same as p = ImVfbQLeft(v,p)

The pixel just to the **ImVfbQRight** of the right-most pixel in a scanline is the left-most pixel in the next scanline down. The pixel just to the **ImVfbQLeft** of the left-most pixel in a scanline is the last pixel in the previous scanline. No automatic wraparound occurs between the last pixel and the first pixel in the frame buffer.

COLOR LOOKUP TABLES

A color lookup table (CLT) is an ordered list of colors. Each color is represented by three 8-bit values. The first value gives the Red component, the second the Green, and the third the Blue component of the color. The triplet is often referred to as an "RGB" value.

Images generally come in two types: pseudo-color and true-color. True-color images store the RGB triplet for each pixel in the image itself. Pseudo-color images store an index into a color table. The RGB triplet for the pixel's color is found at that indexed location in the associated color table.

Allocating a Color Lookup Table

A color lookup table is allocated with a call to

```
ImClt *c;
 ...
c = ImCltAlloc( numColors );
```

where *numColors* indicate how many color entries are to be allocated. Each color entry contains 8 bits of red, 8 bits of green, and 8 bits of blue.

Freeing a Color Lookup Table

A color lookup table's memory can be freed (deallocated) with a call to

```
ImClt *c;
 ...
ImCltFree( c );
```

Freeing a virtual frame buffer with **ImVfbFree** does not free its color lookup table, if it has one.

Per-Color Storage

Information is **Q**ueried from a color lookup table by

Call	Meaning
ImCltQNColors(c)	Queries the number of colors in this **ImClt**
ImCltQRed(p)	Queries the Red component of a **ImClt** entry
ImCltQGreen(p)	Queries the Green component of a **ImClt** entry
ImCltQBlue(p)	Queries the Blue component of a **ImClt** entry

where *c* is the specific color lookup table and *p* is the pointer to a **ImClt** location.

Information is **S**et into a color lookup table by

Call	Meaning
ImCltSRed(p,r)	Sets the Red component of a **ImClt** entry
ImCltSGreen(p,g)	Sets the Green component of a **ImClt** entry
ImCltSBlue(p,b)	Sets the Blue component of a **ImClt** entry

A color lookup table can be attached to a particular virtual frame buffer by

```
ImVfbSClt( v, c ) ;
```

ImVfbQClt(v) can later be used to **Q**uery the pointer to the **ImClt**.

ImClt pointer values can be **Q**ueried and **S**et by

Call	Meaning
ImCltQPtr(c,i)	Returns a pointer to a particular **ImClt** location
ImCltQFirst(c)	Returns a pointer to the first **ImClt** location
ImCltQLast(c)	Returns a pointer to the last **ImClt** location
ImCltQNext(c,p)	Returns a pointer to the next **ImClt** location
ImCltQPrev(c,p)	Returns a pointer to the previous **ImClt** location
ImCltSInc(c,p)	Same as p = ImCltQNext(c,p)
ImCltSDec(c,p)	Same as p = ImCltQPrev(c,p)

IMAGE FILE I/O

Image files are files that contain zero or more images, zero or more color lookup tables, and zero or more other pieces of data such as the image‘s title, date of creation, and so on. Such image files are often the output of proprietary graphics packages, such as Alias or Wavefront, or are graphics vendor standard storage formats, such as Sun’s Rasterfile or Pixar’s PIC file. Many public domain graphics packages also support their own image file formats, such as the Utah Raster Toolkit’s RLE format, or the X Window System’s XWD format.

The **ImFile** routines of the SDSC image library recognize many of these formats and may be used to read an image file into a tag table of virtual frame buffers, color lookup tables, and so on.

Reading an Image File

An image file may be read into a tag table with a call to **ImFileRead** or **ImFileFRead**:

```
ImFileRead( fd, format, flagsTable, dataTable );
ImFileFRead( fp, format, flagsTable, dataTable );
```

where *fd* is a UNIX file descriptor, *fp* is a UNIX **FILE** pointer, *format* is the name of the recognized formats (see **ImFileRead**(3IM) for a list), *flagsTable* is a table of input control flags, and *dataTable* is the table to fill with tag-value pairs of data read in from the file.

For both read calls, the input may be either a file or a pipe.

Writing an Image File

An image file may be written out from a tag table with a call to **ImFileWrite** or **ImFileFWrite**:

```
ImFileWrite( fd, format, flagsTable, dataTable );
ImFileFWrite( fp, format, flagsTable, dataTable );
```

where *fd* is a UNIX file descriptor, *fp* is a UNIX **FILE** pointer, *format* is the name of the recognized formats (see **ImFileWrite**(3IM) for a list), *flagsTable* is a table of output control flags, and *dataTable* is the table of tag-value pairs of data to be written out to the file.

For both write calls, the output may be either a file or a pipe.

NOTES

Frame buffer information can be allocated either on a per-pixel or on a per-plane basis. The per-pixel basis is better to reduce paging. The per-plane allocation is better to reduce overall storage. For a per-pixel allocation, one might call

```
v = ImVfbAlloc( 1280, 1024, IMVFBRGB | IMVFBZ );
```

For a per-plane allocation, one might call

```
vc = ImVfbAlloc( 1280, 1024, IMVFBRGB );
vz = ImVfbAlloc( 1280, 1024, IMVFBZ );
```

RETURNED VALUES

Many of the image library procedures return values that are pointers to a new **ImVfb** or **ImClt**. For Vfb routines, the source Vfb, *sourceVfb*, is used as an input frame buffer. If the value of *destVfb* is equal to **IMVFBNEW**, then an entirely new Vfb will be created and returned. Otherwise, the Vfb indicated by *destVfb* is modified and the value of *destVfb* is returned. *sourceVfb* and *destVfb* can be the same Vfb without negative side effects.

For **ImClt** routines, the source **ImClt**, *sourceClt*, is used as an input lookup table. If the value of *destClt* is equal to **IMCLTNEW**, then an entirely new **ImClt** will be created and returned. Otherwise, the **ImClt** indicated by *destClt* is modified and the value of *destClt* is returned. *sourceClt*, and *destClt* can be the same **ImClt** without negative side effects.

RETURNS

On an error, calls returning pointers to **ImVfb** structures return **IMVFBNULL**, calls returning **ImClt** structures return **IMCLTNULL**, and character string functions return **NULL**. Non-pointer functions return a -1 on errors.

In all cases, on an error, the global **ImErrNo** is set to an error code indicating the cause of the error. A description of the error can be printed with **ImPError** (3IM).

SUMMARY OF IM ROUTINES

char	∗ImFileQFFormat(fp, fileName)

```
    char      *ImFileQFormat( fd, fileName )
    char      *ImQError( )
   ImClt      *ImCltAlloc( numColors )
   ImClt      *ImCltDup( srcClt )
    void      ImCltFree( srcClt )
     int      ImCltQBlue( p )
ImCltPtr      ImCltQFirst( srcClt )
     int      ImCltQGreen( p )
ImCltPtr      ImCltQLast( srcClt )
     int      ImCltQNColors( srcClt )
ImCltPtr      ImCltQNext( srcClt, p )
ImCltPtr      ImCltQPrev( srcClt, p )
ImCltPtr      ImCltQPtr( srcClt, i )
     int      ImCltQRed( p )
    void      ImCltSBlue( p, b )
    void      ImCltSDec( srcClt, p )
    void      ImCltSGreen( p, g )
    void      ImCltSInc( srcClt, p )
    void      ImCltSRed( p, r )
     int      ImFileFRead( fp, "gif", flagsTable, dataTable )
     int      ImFileFRead( fp, "hdf", flagsTable, dataTable )
     int      ImFileFRead( fp, "icon", flagsTable, dataTable )
     int      ImFileFRead( fp, "iff", flagsTable, dataTable )
     int      ImFileFRead( fp, "mpnt", flagsTable, dataTable )
     int      ImFileFRead( fp, "pbm", flagsTable, dataTable )
     int      ImFileFRead( fp, "pcx", flagsTable, dataTable )
     int      ImFileFRead( fp, "pgm", flagsTable, dataTable )
     int      ImFileFRead( fp, "pic", flagsTable, dataTable )
     int      ImFileFRead( fp, "pict", flagsTable, dataTable )
     int      ImFileFRead( fp, "pix", flagsTable, dataTable )
     int      ImFileFRead( fp, "pnm", flagsTable, dataTable )
     int      ImFileFRead( fp, "ppm", flagsTable, dataTable )
     int      ImFileFRead( fp, "ras", flagsTable, dataTable )
     int      ImFileFRead( fp, "rgb", flagsTable, dataTable )
     int      ImFileFRead( fp, "rla", flagsTable, dataTable )
     int      ImFileFRead( fp, "rle", flagsTable, dataTable )
     int      ImFileFRead( fp, "rpbm", flagsTable, dataTable )
     int      ImFileFRead( fp, "rpgm", flagsTable, dataTable )
     int      ImFileFRead( fp, "rpnm", flagsTable, dataTable )
     int      ImFileFRead( fp, "rppm", flagsTable, dataTable )
     int      ImFileFRead( fp, "synu", flagsTable, dataTable )
     int      ImFileFRead( fp, "tiff", flagsTable, dataTable )
     int      ImFileFRead( fp, "x", flagsTable, dataTable )
     int      ImFileFRead( fp, "xbm", flagsTable, dataTable )
     int      ImFileFRead( fp, "xwd", flagsTable, dataTable )
     int      ImFileFRead( fp, format, flagsTable, dataTable )
int     ImFileFWrite( fp, "eps", flagsTable, dataTable )
int     ImFileFWrite( fp, "gif",  flagsTable, dataTable )
int     ImFileFWrite( fp, "hdf", flagsTable, dataTable )
int     ImFileFWrite( fp, "icon", flagsTable, dataTable )
int     ImFileFWrite( fp, "iff", flagsTable, dataTable )
int     ImFileFWrite( fp, "mpnt", flagsTable, dataTable )
int     ImFileFWrite( fp, "pbm", flagsTable, dataTable )
int     ImFileFWrite( fp, "pcx", flagsTable, dataTable )
int     ImFileFWrite( fp, "pgm", flagsTable, dataTable )
int     ImFileFWrite( fp, "pic", flagsTable, dataTable )
int     ImFileFWrite( fp, "pict", flagsTable, dataTable )
int     ImFileFWrite( fp, "pix", flagsTable, dataTable )
int     ImFileFWrite( fp, "pnm", flagsTable, dataTable )
int     ImFileFWrite( fp, "ppm", flagsTable, dataTable )
int     ImFileFWrite( fp, "ps", flagsTable, dataTable )
int     ImFileFWrite( fp, "ras", flagsTable, dataTable )
int     ImFileFWrite( fp, "rgb", flagsTable, dataTable )
int     ImFileFWrite( fp, "rla", flagsTable, dataTable )
int     ImFileFWrite( fp, "rle", flagsTable, dataTable )
int     ImFileFWrite( fp, "rpbm", flagsTable, dataTable )
int     ImFileFWrite( fp, "rpgm", flagsTable, dataTable )
int     ImFileFWrite( fp, "rpnm", flagsTable, dataTable )
int     ImFileFWrite( fp, "rppm", flagsTable, dataTable )
int     ImFileFWrite( fp, "synu", flagsTable, dataTable )
int     ImFileFWrite( fp, "tiff", flagsTable, dataTable )
int     ImFileFWrite( fp, "x", flagsTable, dataTable )
int     ImFileFWrite( fp, "xbm", flagsTable, dataTable )
int     ImFileFWrite( fp, "xwd", flagsTable, dataTable )
int     ImFileFWrite( fp, format, flagsTable, dataTable )
int     ImFileFormatEquivs( baseNEquivs, baseEquivs, totalEquivs )
int     ImFileFormatOptions( baseNOptions, baseOptions, totalOptions )
int     ImFileQNFormat( )
int     ImFileRead( fd, "gif", flagsTable, dataTable )
int     ImFileRead( fd, "hdf", flagsTable, dataTable )
int     ImFileRead( fd, "icon", flagsTable, dataTable )
int     ImFileRead( fd, "iff", flagsTable, dataTable )
int     ImFileRead( fd, "mpnt", flagsTable, dataTable )
int     ImFileRead( fd, "pbm", flagsTable, dataTable )
int     ImFileRead( fd, "pcx", flagsTable, dataTable )
int     ImFileRead( fd, "pgm", flagsTable, dataTable )
int     ImFileRead( fd, "pic", flagsTable, dataTable )
int     ImFileRead( fd, "pict", flagsTable, dataTable )
int     ImFileRead( fd, "pix", flagsTable, dataTable )
int     ImFileRead( fd, "pnm", flagsTable, dataTable )
int     ImFileRead( fd, "ppm", flagsTable, dataTable )
int     ImFileRead( fd, "ras", flagsTable, dataTable )
```

```
     int    ImFileRead( fd, "rgb", flagsTable, dataTable )
     int    ImFileRead( fd, "rla", flagsTable, dataTable )
     int    ImFileRead( fd, "rle", flagsTable, dataTable )
     int    ImFileRead( fd, "rpbm", flagsTable, dataTable )
     int    ImFileRead( fd, "rpgm", flagsTable, dataTable )
     int    ImFileRead( fd, "rpnm", flagsTable, dataTable )
     int    ImFileRead( fd, "rppm", flagsTable, dataTable )
     int    ImFileRead( fd, "synu", flagsTable, dataTable )
     int    ImFileRead( fd, "tiff", flagsTable, dataTable )
     int    ImFileRead( fd, "x", flagsTable, dataTable )
     int    ImFileRead( fd, "xbm", flagsTable, dataTable )
     int    ImFileRead( fd, "xwd", flagsTable, dataTable )
     int    ImFileRead( fd, format, flagsTable, dataTable )
     int    ImFileWrite( fd, "eps", flagsTable, dataTable )
     int    ImFileWrite( fd, "gif",  flagsTable, dataTable )
     int    ImFileWrite( fd, "hdf", flagsTable, dataTable )
     int    ImFileWrite( fd, "icon", flagsTable, dataTable )
     int    ImFileWrite( fd, "iff", flagsTable, dataTable )
     int    ImFileWrite( fd, "mpnt", flagsTable, dataTable )
     int    ImFileWrite( fd, "pbm", flagsTable, dataTable )
     int    ImFileWrite( fd, "pcx", flagsTable, dataTable )
     int    ImFileWrite( fd, "pgm", flagsTable, dataTable )
     int    ImFileWrite( fd, "pic", flagsTable, dataTable )
     int    ImFileWrite( fd, "pict", flagsTable, dataTable )
     int    ImFileWrite( fd, "pix", flagsTable, dataTable )
     int    ImFileWrite( fd, "pnm", flagsTable, dataTable )
     int    ImFileWrite( fd, "ppm", flagsTable, dataTable )
     int    ImFileWrite( fd, "ps", flagsTable, dataTable )
     int    ImFileWrite( fd, "ras", flagsTable, dataTable )
     int    ImFileWrite( fd, "rgb", flagsTable, dataTable )
     int    ImFileWrite( fd, "rla", flagsTable, dataTable )
     int    ImFileWrite( fd, "rle", flagsTable, dataTable )
     int    ImFileWrite( fd, "rpbm", flagsTable, dataTable )
     int    ImFileWrite( fd, "rpgm", flagsTable, dataTable )
     int    ImFileWrite( fd, "rpnm", flagsTable, dataTable )
     int    ImFileWrite( fd, "rppm", flagsTable, dataTable )
     int    ImFileWrite( fd, "synu", flagsTable, dataTable )
     int    ImFileWrite( fd, "tiff", flagsTable, dataTable )
     int    ImFileWrite( fd, "x", flagsTable, dataTable )
     int    ImFileWrite( fd, "xbm", flagsTable, dataTable )
     int    ImFileWrite( fd, "xwd", flagsTable, dataTable )
     int    ImFileWrite( fd, format, flagsTable, dataTable )
    void    ImPError( str )
   ImVfb    *ImVfbAlloc( width, height, fields )
   ImVfb    *ImVfbCopy( srcVfb, srcXLeft, srcYTop, srcDX, srcDY, fieldMask, dstVfb, dstXLeft, dstYTop )
   ImVfb    *ImVfbDup( srcVfb )
```

```
     ImVfb   *ImVfbFlip( srcVfb, flipDirection, dstVfb )
      void   ImVfbFree( srcVfb )
       int   ImVfbQAlpha( srcVfb, p )
       int   ImVfbQBlue( srcVfb, p )
     ImClt   *ImVfbQClt( srcVfb )
  ImVfbPtr   ImVfbQDown( srcVfb, p )
     float   ImVfbQFData( srcVfb, p )
       int   ImVfbQFields( srcVfb )
  ImVfbPtr   ImVfbQFirst( srcVfb )
       int   ImVfbQGray( srcVfb, p )
       int   ImVfbQGreen( srcVfb, p )
       int   ImVfbQGrey( srcVfb, p )
       int   ImVfbQHeight( srcVfb )
       int   ImVfbQIData( srcVfb, p )
       int   ImVfbQIndex( srcVfb, p )
       int   ImVfbQIndex16( srcVfb, p )
       int   ImVfbQIndex8( srcVfb, p )
  ImVfbPtr   ImVfbQLast( srcVfb )
  ImVfbPtr   ImVfbQLeft( srcVfb, p )
       int   ImVfbQNBytes( srcVfb )
  ImVfbPtr   ImVfbQNext( srcVfb, p )
  ImVfbPtr   ImVfbQPrev( srcVfb, p )
  ImVfbPtr   ImVfbQPtr( srcVfb, x, y )
       int   ImVfbQRed( srcVfb, p )
  ImVfbPtr   ImVfbQRight( srcVfb, p )
  ImVfbPtr   ImVfbQUp( srcVfb, p )
       int   ImVfbQWidth( srcVfb )
       int   ImVfbQZ( srcVfb, p )
     ImVfb   *ImVfbResize( srcVfb, algorithm, dstVfb, width, height )
      void   ImVfbSAlpha( srcVfb, p, a )
      void   ImVfbSBlue( srcVfb, p, b )
      void   ImVfbSClt( srcVfb, clt )
      void   ImVfbSDec( srcVfb, p )
      void   ImVfbSFData( srcVfb, p, f )
      void   ImVfbSGray( srcVfb, p, g )
      void   ImVfbSGreen( srcVfb, p, g )
      void   ImVfbSGrey( srcVfb, p, g )
      void   ImVfbSIData( srcVfb, p, i )
      void   ImVfbSInc( srcVfb, p )
      void   ImVfbSIndex( srcVfb, p, i )
      void   ImVfbSIndex16( srcVfb, p, i32 )
      void   ImVfbSIndex8( srcVfb, p, i8 )
      void   ImVfbSRed( srcVfb, p, r )
      void   ImVfbSZ( srcVfb, p, z )
     ImVfb   *ImVfbToGray( srcVfb, dstVfb )
     ImVfb   *ImVfbToGrey( srcVfb, dstVfb )
```

```
ImVfb      *ImVfbToIndex16( srcVfb, dstVfb )
ImVfb      *ImVfbToIndex8( srcVfb, dstVfb )
ImVfb      *ImVfbToMono( srcVfb, threshold, dstVfb )
ImVfb      *ImVfbToRgb( srcVfb, dstVfb )
```

AUTHORS

Mike Bailey, T. Todd Elvins, and Dave Nadeau,
with Don Doering, Jesus Ferrer, Soraya Gonzalez, Jim McLeod, Phil Mercurio, John Moreland,
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImCltAlloc - allocate a color lookup table

SYNOPSIS

```
#include "im.h"

ImClt ∗ImCltAlloc( numColors )
        int numColors ;
```

DESCRIPTION

ImCltAlloc allocates a color lookup table with *numColors* entries and returns a pointer to the new **ImClt**. Each color lookup table entry represents a color using 8 bits of red, 8 bits of green, and 8 bits of blue.

The new color table is not initialized (colors will be random).

NOTES

Color lookup table entries are accessed by obtaining a pointer to a specific entry using

Macro	Meaning
ImCltQPtr(c, i)	Returns a pointer to a particular **ImClt** entry
ImCltQFirst(c)	Returns a pointer to the first **ImClt** entry
ImCltQLast(c)	Returns a pointer to the last **ImClt** entry
ImCltQNext(c, p)	Returns a pointer to the next **ImClt** entry
ImCltQPrev(c, p)	Returns a pointer to the previous **ImClt** entry
ImCltSInc(c, p)	Same as **p = ImCltQNext(c, p)**
ImCltSDec(c, p)	Same as **p = ImCltQPrev(c, p)**

where *c* is the specific color lookup table, *i* is an entry number (first entry is entry number 0), and *p* is a pointer to a **ImClt** entry.

Using an **ImClt** entry pointer, entries in a color lookup table may be set and queried using:

Macro	Meaning
ImCltSRed(p, r)	Sets red component of a **ImClt** location
ImCltQRed(p)	Queries the red component of a **ImClt** location
ImCltSGreen(p, g)	Sets green component of a **ImClt** location
ImCltQGreen(p)	Queries the green component of a **ImClt** location
ImCltSBlue(p, b)	Sets blue component of a **ImClt** location
ImCltQBlue(p)	Queries the blue component of a **ImClt** location

The number of entries in a color lookup table may queried using:

Macro	Meaning
ImCltQNColors(c)	Queries the number of colors in this **ImClt**

A color lookup table may be attached to a virtual frame buffer (see **ImVfbAlloc**(3IM)). To attach (set) the virtual frame buffer's color lookup table, use **ImVfbSClt**(3IM). To query a virtual frame buffer's currently attached color lookup table, use **ImVfbQClt**(3IM).

Macro	Meaning
ImVfbSClt(v, c)	Attaches (set) the **ImClt** for an **ImVfb**
ImVfbQClt(v)	Queries the **ImClt** for an **ImVfb**

RETURNS

Upon success, **ImCltAlloc** returns a new **ImClt**. On failure, **IMCLTNULL** is returned and **ImErrNo** set to the following:

IMEMALLOC	Cannot allocate enough memory for the new **ImClt**

SEE ALSO

ImIntro (3IM), **ImErrNo** (3IM), **ImCltDup** (3IM), **ImCltFree** (3IM), **ImCltGrayRamp** (3IM), **ImCltQBlue** (3IM), **ImCltQFirst** (3IM), **ImCltQGreen** (3IM), **ImCltQLast** (3IM), **ImCltQNColors** (3IM), **ImCltQNext** (3IM), **ImCltQPrev** (3IM), **ImCltQPtr** (3IM), **ImCltQRed** (3IM), **ImCltSBlue** (3IM), **ImCltSDec** (3IM), **ImCltSGreen** (3IM), **ImCltSInc** (3IM), **ImCltSRed** (3IM), **ImVfbQClt** (3IM), **ImVfbSClt** (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

ImCltDup - duplicate a color lookup table

SYNOPSIS

```
#include "im.h"

ImClt *ImCltDup( srcClt )
        ImClt *srcClt ;
```

DESCRIPTION

ImCltDup duplicates a color lookup table *srcClt* and returns a pointer to the new **ImClt**. The *srcClt* is not changed by this operation.

RETURNS

Upon success, **ImCltDup** returns a new **ImClt**. On failure, **IMCLTNULL** is returned and **ImErrNo** set to one of the following:

IMEMALLOC	Cannot allocate enough memory for the new **ImClt**

SEE ALSO

ImIntro (3IM), **ImErrNo** (3IM), **ImCltAlloc** (3IM), **ImCltFree** (3IM), **ImCltGrayRamp** (3IM), **ImCltQBlue** (3IM), **ImCltQFirst** (3IM), **ImCltQGreen** (3IM), **ImCltQLast** (3IM), **ImCltQNColors** (3IM), **ImCltQNext** (3IM), **ImCltQPrev** (3IM), **ImCltQPtr** (3IM), **ImCltQRed** (3IM), **ImCltSBlue** (3IM), **ImCltSDec** (3IM), **ImCltSGreen** (3IM), **ImCltSInc** (3IM), **ImCltSRed** (3IM), **ImVfbQClt** (3IM), **ImVfbSClt** (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImCltFree - free the storage for a color lookup table

SYNOPSIS

#include "im.h"

void ImCltFree(srcClt)
ImClt ∗srcClt ;

DESCRIPTION

ImCltFree frees the dynamic storage for color lookup table *srcClt*. *srcClt* is no longer valid after this call and should not be used.

NOTES

It is assumed that *srcClt* was originally allocated with **ImCltAlloc** (3IM).

If *srcClt* is attached to a virtual frame buffer (see **ImVfbSClt** (3IM)), it should be freed using **ImCltFree** prior to freeing the virtual frame buffer using **ImVfbFree** (3IM).

RETURNS

ImCltFree returns nothing.

SEE ALSO

ImIntro (3IM), **ImErrNo** (3IM), **ImCltAlloc** (3IM), **ImCltDup** (3IM), **ImCltGrayRamp** (3IM), **ImCltQBlue** (3IM), **ImCltQFirst** (3IM), **ImCltQGreen** (3IM), **ImCltQLast** (3IM), **ImCltQNColors** (3IM), **ImCltQNext** (3IM), **ImCltQPrev** (3IM), **ImCltQPtr** (3IM), **ImCltQRed** (3IM), **ImCltSBlue** (3IM), **ImCltSDec** (3IM), **ImCltSGreen** (3IM), **ImCltSInc** (3IM), **ImCltSRed** (3IM), **ImVfbQClt** (3IM), **ImVfbSClt** (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImCltQNColors - query the number of colors in a color lookup table

SYNOPSIS

```
#include "im.h"

int ImCltQNColors( srcClt )
        ImClt *srcClt ;
```

DESCRIPTION

ImCltQNColors queries the number of color entries within *srcClt*.

NOTES

ImCltQNColors is a C macro.

RETURNS

ImCltQNColors returns the number of entries in the color lookup table.

SEE ALSO

ImIntro (3IM), **ImErrNo** (3IM), **ImCltAlloc** (3IM), **ImCltDup** (3IM), **ImCltFree** (3IM), **ImCltGrayRamp** (3IM), **ImCltQBlue** (3IM), **ImCltQFirst** (3IM), **ImCltQGreen** (3IM), **ImCltQLast** (3IM), **ImCltQNext** (3IM), **ImCltQPrev** (3IM), **ImCltQPtr** (3IM), **ImCltQRed** (3IM), **ImCltSBlue** (3IM), **ImCltSDec** (3IM), **ImCltSGreen** (3IM), **ImCltSInc** (3IM), **ImCltSRed** (3IM), **ImVfbQClt** (3IM), **ImVfbSClt** (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImCltQPtr - query the pointer to a particular entry in a color lookup table
ImCltQFirst - query the pointer to the first entry in a color lookup table
ImCltQLast - query the pointer to the last entry in a color lookup table
ImCltQNext - query a pointer to the next entry in a color lookup table
ImCltQPrev - query a pointer to the previous entry in a color lookup table
ImCltSInc - increment a pointer to point to the next entry in a color lookup table
ImCltSDec - decrement a pointer to point to the previous entry in a color lookup table

SYNOPSIS

```
#include "im.h"

ImCltPtr ImCltQPtr( srcClt, i )
        ImClt *srcClt ;
        int i ;

ImCltPtr ImCltQFirst( srcClt )
        ImClt *srcClt ;

ImCltPtr ImCltQLast( srcClt )
        ImClt *srcClt ;

ImCltPtr ImCltQNext( srcClt, p )
        ImClt *srcClt ;
        ImCltPtr p ;

ImCltPtr ImCltQPrev( srcClt, p )
        ImClt *srcClt ;
        ImCltPtr p ;

void ImCltSInc( srcClt, p )
        ImClt *srcClt ;
        ImCltPtr p ;

void ImCltSDec( srcClt, p )
        ImClt *srcClt ;
        ImCltPtr p ;
```

DESCRIPTION

These macros query and set pointers to color lookup table entries.

ImCltQPtr returns a pointer to the *i*-th entry in *srcClt*. Entries are numbered from 0 for the first entry.

ImCltQFirst returns a pointer to the first entry in *srcClt* and is a synonym for **ImCltQPtr(srcClt, 0)**.

ImCltQLast returns a pointer to the last entry in *srcClt* and is a synonym for **ImCltQPtr(srcClt, ImCltQNColors(srcClt) - 1)**.

ImCltQNext returns a pointer to the *srcClt* entry following *p*.

ImCltQPrev returns a pointer to the *srcClt* entry preceding *p*.

ImCltSInc increments the pointer *p* to advance it to the next entry in *srcClt*. This is the same as **p = ImCltQNext(srcClt, p)**.

ImCltSDec decrements the pointer *p* to advance it to the previous entry in *srcClt*. This is the same as **p = ImCltQPrev(srcClt, p)**.

NOTES

Each of these are C macros.

RETURNS

ImCltQPtr, **ImCltQFirst**, **ImCltQLast**, **ImCltQNext**, and **ImCltQPrev** return a pointer to a color lookup table entry.

ImCltSInc and **ImCltSDec** return nothing.

SEE ALSO

ImIntro (3IM), **ImErrNo** (3IM), **ImCltAlloc** (3IM), **ImCltDup** (3IM), **ImCltFree** (3IM), **ImCltGrayRamp** (3IM), **ImCltQBlue** (3IM), **ImCltQGreen** (3IM), **ImCltQNColors** (3IM), **ImCltQRed** (3IM), **ImCltSBlue** (3IM), **ImCltSGreen** (3IM), **ImCltSRed** (3IM), **ImVfbQClt** (3IM), **ImVfbSClt** (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

ImCltQRed - query the red component of a color in a color lookup table
ImCltQGreen - query the green component of a color in a color lookup table
ImCltQBlue - query the blue component of a color in a color lookup table
ImCltSRed - set the red component of a color in a color lookup table
ImCltSGreen - set the green component of a color in a color lookup table
ImCltSBlue - set the blue component of a color in a color lookup table

SYNOPSIS

```
#include "im.h"

int ImCltQRed( p )
        ImCltPtr p ;

int ImCltQGreen( p )
        ImCltPtr p ;

int ImCltQBlue( p )
        ImCltPtr p ;

void ImCltSRed( p, r )
        ImCltPtr p ;
        int r ;

void ImCltSGreen( p, g )
        ImCltPtr p ;
        int g ;

void ImCltSBlue( p, b )
        ImCltPtr p ;
        int b ;
```

DESCRIPTION

These macros set and query color entry information for a color lookup table.

ImCltQRed, **ImVfbQGreen**, and **ImVfbQBlue** query the red, green, and blue components of a color lookup table entry pointed to by *p*. The specific 8-bit color component is returned as the function's value.

ImCltSRed, **ImVfbSGreen**, and **ImVfbSBlue** set the red, green, and blue components of a color lookup table entry pointed to by *p*. *r*, *g*, and *b* arguments are the 8-bit values to use to set the color component.

NOTES

Color lookup table entry pointers may be obtained using

Macro	Meaning
ImCltQPtr(c, i)	Returns a pointer to a particular **ImClt** entry
ImCltQFirst(c)	Returns a pointer to the first **ImClt** entry

ImCltQLast(c)	Returns a pointer to the last **ImClt** entry
ImCltQNext(c, p)	Returns a pointer to the next **ImClt** entry
ImCltQPrev(c, p)	Returns a pointer to the previous **ImClt** entry
ImCltSInc(c, p)	Same as **p = ImCltQNext(c, p)**
ImCltSDec(c, p)	Same as **p = ImCltQPrev(c, p)**

where *c* is the specific color lookup table, *i* is an entry number (first entry is entry number 0), and *p* is a pointer to a **ImClt** entry.

Each of these are C macros.

RETURNS

ImCltQRed, **ImVfbQGreen**, and **ImVfbQBlue** each return an integer containing, in its lowest 8 bits, the color component queried.

ImCltSRed, **ImVfbSGreen**, and **ImVfbSBlue** return nothing.

SEE ALSO

ImIntro (3IM), **ImErrNo** (3IM), **ImCltAlloc** (3IM), **ImCltDup** (3IM), **ImCltFree** (3IM), **ImCltGrayRamp** (3IM), **ImCltQFirst** (3IM), **ImCltQLast** (3IM), **ImCltQNColors** (3IM), **ImCltQNext** (3IM), **ImCltQPrev** (3IM), **ImCltQPtr** (3IM), **ImCltSDec** (3IM), **ImCltSInc** (3IM), **ImVfbQClt** (3IM), **ImVfbSClt** (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

imeps - SDSC Encapsulated PostScript file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileWrite( fd, "eps", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "eps", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

Encapsulated PostScript color, grayscale, and black-and-white image files are used by a variety of PostScript tools in order to include PostScript diagrams and images within other documents.

FILE RECOGNITION

Encapsulated PostScript files are recognized by the following filename suffixes: .eps, .epi, .epsi, and .epsf.

NOTES

SDSC image library support of Encapsulated PostScript does not require use of any windowing system libraries or PostScript interpreters, and contains no proprietary code. Encapsulated PostScript format handling is available on any machine for which the SDSC image library is available.

PostScript is a reverse polish notation-style, FORTH-like programming language used to describe text and pictures to be rendered by a PostScript interpreter embedded within a laser printer (such as Apple's LaserWriter printers), windowing system (such as Sun's NeWS or NeXT's NeXTstep), or other display tool. The language contains numerous basic programming language constructs and lots of drawing operations.

PostScript was designed for ease of parsing, not ease of programming. As a result, most people use PostScript as a "blackbox" file format that describes their picture. It is left up to software, such as the SDSC image library, to figure out how to deal with PostScript.

Encapsulated PostScript (EPS) is a file format for the import and export of PostScript language files among applications. EPS files contain a PostScript program fragment that draws a diagram or renders an image when printed to a PostScript printer or displayed in a windowing system using a PostScript interpreter application. EPS files are intended to be used by applications that wish to embed complex PostScript drawings within non-PostScript data, such as a word processing document, or a spreadsheet.

EPS files may contain an optional grayscale "Preview image" of the same size and shape as the image rendered by the main PostScript program in the file. The "Preview image" is used by non-PostScript applications to display to the user a rough approximation of the EPS file's image. The user may position, orient, and scale the "Preview image" within the including document. When the including application prints the full document, the EPS file's contents are substituted for the "Preview image". The user's position, orient, and scale operations are applied to the EPS data and the completed PostScript file sent to the printer.

The EPS "Preview image" may be provided in a variety of ways:

not given	Simple EPS file. No "Preview image".
DVI image	Device-independent EPS file. Grayscale bitmap "Preview image" in header.
Metafile image	IBM PC EPS file. MS Windows Metafile "Preview image" in header.
PICT image	Mac EPS file. PICT "Preview image" in resource fork.
TIFF image	Portable EPS file. TIFF "Preview image" in header.

EPS files with a device-independent bitmap as a "Preview image" are generally refered to as EPSI or EPI files. Each of the other EPS file types are known as EPSF files.

Mac EPS files include a PICT image in the resource fork of the file. Since non-Mac systems do not support the Mac notion of data and resource forks, it is not possible to generate Mac EPS files on anything but a Mac.

IBM PC EPS files include either a TIFF or MS Windows Metafile image in the header of the file. However, inclusion of either image type prevents the EPS file from being sent directly to a PostScript laser printer for a quick printout. IBM PC EPS files must first be included in a document and stripped of their "Preview image" headers before they may be printed.

Device-independent EPS files are straight PostScript. The EPSI "Preview image" is included as a PostScript comment in the file’s header. This allows EPSI files to be sent directly to a PostScript laser printer without any extra processing.

Encapsulated PostScript support within the SDSC Image Library generates EPSI (device-independent) files.

Reading Encapsulated PostScript files

Reading of Encapsulated PostScript is not supported by the SDSC image library.

Support for EPS file reading would require one of two approaches: 1. interpret the EPS file’s PostScript, or 2. interpret the "Preview image" in the header.

Inclusion of a full PostScript language interpreter in order to read an EPS file would require a great deal of code. This approach is not really practical.

Users needing to read in PostScript data should investigate PostScript interpreters, such as Sun Microsystem’s NeWS window system and its image viewer **PageView**(1), or GNU’s **GostScript** PostScript-clone interpreter.

Interpretation of an EPSI file’s "Preview image" is not of much use within an imaging tool set. EPSI file "Preview images" are strictly grayscale, regardless of possible color content in the EPSI file’s actual PostScript image. To claim to read EPSI files, yet only read the crude grayscale "Preview image" would be inappropriate.

Writing Encapsulated PostScript files

The SDSC image library writes color, grayscale, or monochrome VFBs as EPSI Encapsulated PostScript files. In each case the generated Encapsulated PostScript consists of a header followed by the image data as ASCII hex numbers. The choice of header and the format of the hex numbers depends upon the type of image being written.

IMVFBMONO VFBs are written with an Encapsulated PostScript header that renders the image in black-and-white on black-and-white or color devices. Image data is written as two hex characters for each group of eight adjacent monochrome pixels (as required by the PostScript image operator).

IMVFBINDEX8 VFBs without color lookup tables are written with a Encapsulated PostScript header that renders the image in shades of gray on black-and-white or color devices. Image data is written as two hex characters for each grayscale pixel (as required by the PostScript image operator).

IMVFBINDEX8 VFBs with a color lookup table and **IMVFBRGB** VFBs are written with an Encapsulated PostScript header that renders the image in color on color devices and in shades of gray on black-and-white devices. The PostScript header code checks the device's systemdict for support of the colorimage operator. If the operator exists, the device supports color. In both cases, image data is written as six hex characters at two characters each for the red, green, and blue color component of each pixel (as required by the PostScript colorimage operator). On color devices, this color pixel data generates a color image. On monochrome devices, the header code automatically converts the color pixel data to shades of gray as it is being rendered. This allows the same color PostScript file to be sent to color or black-and-white printers without any change to the file.

Other image library VFB types are converted to one of the above prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

Image data is always generated for pixels in left to right order, from the top of the image to the bottom.

In all cases a grayscale "Preview image" comment is placed at the top of the file.

Encapsulated PostScript generated by the SDSC image library conforms to Adobe's version 3.0 document structuring conventions.

Warning: All PostScript laser printers have a fixed amount of memory in which to buffer incoming data and build up the image prior to printing. Large images converted to PostScript and then sent to a laser printer may exceed the buffering and image-building memory limitations of the printer. Results vary from printer to printer: some lock up and require a reset, while others silently ignore the images and do not print anything.

ERRORS

In addition to those listed for **ImFileWrite**(3IM), Encapsulated PostScript file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

PostScript Language - Tutorial and Cookbook, Adobe Systems Incorporated.

PostScript Language - Reference Manual, Second Edition, Adobe Systems Incorporated.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHORS

Dave Nadeau
San Diego Supercomputer Center

Loren "Buck" Buchanan
Naval Research Laboratory
Kestrel Associates, Inc.

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

ImFileFormatOptions - add file format names to an ArgOption list
ImFileFormatEquivs - add file format equivalent names to an ArgEquiv list

SYNOPSIS

```
#include "im.h"
#include "arg.h"

int ImFileFormatOptions( baseNOptions, baseOptions, totalOptions )
        int baseNOptions;
        ArgOption *baseOptions;
        ArgOption **totalOptions;

int ImFileFormatEquivs( baseNEquivs, baseEquivs, totalEquivs )
        int baseNOEquivs;
        ArgEquiv *baseEquivs;
        ArgEquiv **totalEquivs;
```

DESCRIPTION

These functions are used to help set up argument parsing option and equivalent keyword lists for use with **ArgParse**(3ARG).

ImFileFormatOptions creates a new list of **ArgOption** structures containing the **baseNOptions** basic options in *baseOptions*, plus one option for each image file format supported by the image library. Each file format option entry has the following attributes:

arg_keyword	= file format's name
arg_valuenames	= NULL
arg_help	= format's help information
arg_flags	= **ARGFMULTIPLE**
arg_minvalues	= 0
arg_maxvalues	= 0
arg_type	= **ARGTNONE**

The new option list is returned in *totalOptions*, and the number of options in that list returned as the function's value.

ImFileFormatEquivs creates a new list of **ArgEquiv** structures containing the *baseNEquivs* basic equivalent keywords in *baseEquivs*, plus one equivalent keyword entry for each equivalent file format name for each image file format supported by the image library. Each new equivalent keyword entry is equivalenced to the name of the image file format.

The new equivalent keyword list is returned in *totalEquivs*, and the number of equivalent keywords in that list returned as the function value.

NOTES

Information on the SDSC argument parsing package and its data structures is in **ArgIntro**(3ARG). We assume the user is familiar with the package.

Tools that have one command-line option per image file format name use these routines. Such options typically select the type of image file to read in or write out (see **imconv**(1IM)).

To keep image library tools uniform and consistent, we suggest tools that use **ImFileFormatOptions** also use **ImFileFormatEquivs** to add the equivalent file format names to the option set as well.

ImFileFormatEquivs format equivalences assume that there is one keyword per format and that it is the format name (as done by **ImFileFormatOptions**).

EXAMPLES

The following code declares and initializes an **ArgOption** array with options for a mythical tool **immyth**. Within **main**, **ImFileFormatOptions** and **ImFileFormatEquivs** are called to create new **ArgOption** and **ArgEquiv** arrays that incorporate both the basic **immyth** options and the image file format name options. These new arrays are then passed to **ArgParse**(3ARG) as complete descriptions of the argument set available to the **immyth**.

```
#include "im.h"
#include "arg.h"

/* Describe the command. */
ArgCommand mythCommand =
{
      "immyth", 1, 0, 0,
      "%command is a demo program that does nothing.",
      NULL,
      NULL, NULL,
      ARGFNONE,
      NULL, NULL,
      "SDSC Image Tools, October 1991.",
      "Copyright (c) 1989-1991  San Diego Supercomputer Center (SDSC), CA, USA",
      NULL, NULL
};

/* Describe the command-specific options. */
ArgOption mythBaseOptions[ ] =
{
      { "infile", "image_filename", "Specify an input image file name",
      ARGFREQUIRED | ARGFIMPKEYWORD, 1, 1, ARGTSTRING },

      { "outfile", "image_filename", "Specify an output image file name",
      ARGFREQUIRED | ARGFIMPKEYWORD, 1, 1, ARGTSTRING },

      { "debug", NULL, "Enable debug mode",
      ARGFHIDDEN, 0, 0, ARGTNONE }
};
#define MYTHNBASEOPTIONS 3

/* Describe the command-specific equivalent keywords. */
ArgEquiv mythBaseEquivs[ ] =
{
      { "debug", "dbg" }
};
#define MYTHNBASEEQUIVS 1
```

```
main( argc, argv )
        int argc;
        char *argv[ ];
{
        ArgOption *totalOptions;
        ArgEquiv *totalEquivs;
        int totalNOptions;
        int totalNEquivs;

        /* Add image file names to option set. */
        totalNOptions = ImFileFormatOptions( MYTHNBASEOPTIONS, mythBaseOptions,
                &totalOptions );

        /* Add image file equivalent names to equivalence set. */
        totalNEquivs = ImFileFormatEquivs( MYTHNBASEEQUIVS, mythBaseEquivs,
                &totalEquivs );

        /* Parse the arguments. */
        ArgParse( argc, argv, &mythCommand,
                totalNOptions, totalOptions,
                totalNEquivs, totalEquivs );

        /* Do work. */
        ...
}
```

RETURNS

On success, **ImFileFormatOptions** returns the number of **ArgOption** structures in the new *totalOptions* list. On failure, a -1 is returned and **ImErrNo** set to

IMEMALLOC	Cannot allocate memory

On success, **ImFileFormatEquivs** returns the number of **ArgEquiv** structures in the new *totalEquivs* list. On failure, a -1 is returned and **ImErrNo** set to

IMEMALLOC	Cannot allocate memory

SEE ALSO

ImIntro (3IM), **ImFileRead** (3IM), **ImFileWrite** (3IM) **ArgIntro** (3ARG), **ArgParse** (3ARG)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImFileQFormat, ImFileQFFormat - query file’s image format

SYNOPSIS

```
#include "im.h"

char *ImFileQFormat( fd, fileName )
        int fd;
        char *fileName;

char *ImFileQFFormat( fp, fileName )
        FILE *fp;
        char *fileName;
```

DESCRIPTION

ImFileQFormat and **ImFileQFFormat** determine the image file format in use by the stream selected by *fd* or *fp*, respectively. The name of the format is returned.

NOTES

If the stream is a pipe or device, or a file opened for writing, the stream is left untouched with no data read in. The input’s image file format is determined by matching the *fileName* extension (characters after the last "." in name) against a list of formats supported by the image library. If no match is found, a NULL character string is returned by both functions.

If the stream is a file opened for reading, data is read in and checked against a list of magic numbers for formats supported by the image library. If no match is found, the *fileName* extension is checked, as above. If no match is found here either, a NULL character string is returned by both functions. In any case, the stream is reset to point to the first byte of the file.

If a match is found, a character pointer to the name of the matched image file format is returned as the function’s value. The returned pointer points to internal table space that should not be modified by the user.

RETURNS

Upon success, the name of the format used by the stream is returned. On failure, a NULL character pointer is returned and IImErrNo set to

IMESYS	System call error
IMEFORMAT	Unknown image file format

SEE ALSO

ImIntro (3IM), **ImFileRead** (3IM), **ImFileWrite** (3IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImFileQNFormat - query number of supported image file formats

SYNOPSIS

#include "im.h"

int ImFileQNFormat( )

DESCRIPTION

ImFileQNFormat returns the number of image file formats supported by the image library.

NOTES

ImFileQNFormat is typically called to obtain a loop count when searching the image library's **ImFileFormats** file format information table.

RETURNS

The number of formats in the **ImFileFormats** table is returned. If there are no formats in the table, 0 is returned.

SEE ALSO

ImIntro (3IM), **ImFileRead** (3IM), **ImFileWrite** (3IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImFileRead, ImFileFRead - Read an image format input stream into a Vfb
ImFileWrite, ImFileFWrite - Write a Vfb onto an image format output stream

SYNOPSIS

```
#include "im.h"

int ImFileRead( fd, format, flagsTable, dataTable )
        int fd;
        char *format;
        TagTable *flagsTable;
        TagTable *dataTable;

int ImFileFRead( fp, format, flagsTable, dataTable )
        FILE *fp;
        char *format;
        TagTable *flagsTable;
        TagTable *dataTable;

int ImFileWrite( fd, format, flagsTable, dataTable )
        int fd;
        char *format;
        TagTable *flagsTable;
        TagTable *dataTable;

int ImFileFWrite( fp, format, flagsTable, dataTable )
        FILE *fp;
        char *format;
        TagTable *flagsTable;
        TagTable *dataTable;
```

DESCRIPTION

ImFileRead and **ImFileFRead** read image data of type *format* from the input stream selected by *fd* or *fp*, respectively. Format-specific flags in *flagsTable* may modify the behavior of the read operation. Data read from the stream is appended to *dataTable*.

ImFileWrite and **ImFileFWrite** write image data of type *format* onto the output stream selected by *fd* or *fp*, respectively. Format-specific flags in *flagsTable* may modify the behavior of the write operation. Data to write is taken from the *dataTable*.

NOTES

The image file I/O routines handle the input and output of image data in a variety of image file formats. Read and write routines read or write using file descriptors (unbuffered I/O) or file pointers (buffered I/O), allowing both files and pipes to be handled identically.

Image Formats

Format names are NULL-terminated character strings giving the name, or any of the equivalent names for any supported image file format. The list of image file formats supported is ever-growing. As of this writing, the format list includes the following:

Primary Name	Equivalent Names	Description
gif	giff	CompuServe Graphics Image Format file
hdf	df, ncsa	Hierarchical Data Format file
icon	cursor, pr	Sun Icon and Cursor file
iff	suniff, taac	Sun TAAC Image File Format
mpnt	macp, pntg	Apple Macintosh MacPaint file
pbm	-	Portable Bit Map file
pcx	pcc	ZSoft PC Paintbrush file
pgm	-	Portable Grayscale Map file
pic	picio, pixar	PIXAR PICture file
pict	pict2	Apple Macintosh QuickDraw/PICT picture file
pix	alias	Alias PIXel image file
pnm	-	Portable aNy Map file
ppm	-	Portable Pixel Map file
ps	postscript	PostScript image file
ras	sun, sr, scr	Sun RASterfile
rgb	iris, sgi	Silicon Graphics RGB image file
rla	rlb	Wavefront raster image file
rle	-	Utah Run-Length-Encoded image file
rpbm	-	Raw Portable Bit Map file
rpgm	-	Raw Portable Grayscale Map file
rpnm	-	Raw Portable aNy Map file
rppm	-	Raw Portable Pixel Map file
synu	-	Synu image file
tiff	tif	Tagged Image File
x	avs	Stardent AVS X image file
xbm	bm	X11 Bit Map file
xwd	x11	X11 Window Dump image file

For a current list of image format names supported by the library, execute the **imformats**(1IM) command, or scan the global **ImFileFormats** table of format information.

Individual **man** pages are available for each of the above file formats. The naming convention is primary name with *im* prefix. For instance, the **man** page on image library support of the Sun RASterfile format is **imras**(3IM).

Data Tag Table Entries

Image file data read in by **ImFileRead** or **ImFileFRead** is appended to the *dataTable* in the order in which it is encountered in the file. Each piece of information is given its own table entry and tagged with a standard or custom tag name. Standard tags are given below. Custom tags are format-specific and are discussed in the individual format **man** pages.

Data written by **ImFileWrite** or **ImFileFWrite** is written in the same order as tags encountered in the *dataTable*. Table entries with tags applicable to the format are used, while the rest are ignored.

The following standard *dataTable* tags are recognized by most file format read and write handlers:

image vfb

ImVfb pointer to an image.

When reading multi-image files, each image is given its own entry in the *dataTable,* in the same order as the images occur in the file.

When writing multi-image *dataTable*s, images are written to the file in the same order as they occur in the *dataTable*. If the format does not support multiple images per file, the format handler returns an error and does not write the file.

image clt

ImClt pointer to a color lookup table.

When reading a color lookup table (CLT) and image from a file, both the CLT and the image are listed in the *dataTable*; the CLT first and the image second. If it is clear from the format's specification that the CLT is to be associated with the image, the CLT also is attached to the image virtual frame buffer using **ImVfbSClt**(3IM).

When reading multi-image files, the CLT (if any) for each image immediately precedes the image when placed into the *dataTable.*

When writing a CLT and image from a *dataTable*, the CLT (if any) associated with the image virtual frame buffer takes precedence over any in the *dataTable*. **image clt** tags are ignored.

Format Flags

The *flagsTable* argument on read and write calls may contain flags that select format-specific options. A *flagsTable* flag may request a particular image compression scheme, may block output of an image's CLT, may request that the image be output without its alpha channel, and so on. In all cases, *flagsTable* entries direct format output handlers **how** to output information. **What** to output is the domain of the *dataTable*.

A NULL pointer for the *flagsTable* indicates no flags are provided.

Standard *flagsTable* tags are given below. Custom format-specific tags are discussed in the individual format man pages. Examples and interdependency explanations follow in later sections of this **man** page.

error handler

Integer function pointer for the function to be called and passed an error message. If not given, there is no error handler.

See below for usage of the error handler.

error stream

FILE pointer for the stream to which error messages should be output, such as **stderr**. If not given, there is no error stream. See below for usage of the error stream.

file name

Character string name of the input or output file. If not given, the file name defaults to **file**. If input or output is to a stream instead of a file, the strings **stdin** or **stdout** should be used.

File name is used when constructing error message text to be printed to the error stream and passed to the error handler (see below). File name has no other significance and is not opened directly.

image alpha request

Integer flag to block or force an alpha plane to be output. If not given, an alpha plane is output if the image has one and the format can support it. If the tag forces output when a file format cannot support it or blocks output when a format must have it, an error is returned.

This tag is only used by image write handlers. A value of **IMALPHADUMP** forces alpha plane output. A value of **IMALPHANODUMP** blocks output.

image channel number request

Integer number of image channels to output. If not given, each channel in each incoming image is output or the closest number supported by the output file format. If this tag specifies a number of channels that cannot be supported by the image file format, an error is returned.

This tag is only used by image write handlers.

Channel numbers are typically 1 or 3. Channel image number 1 corresponds to a monochrome, grayscale, or color index image where 1 integer per pixel is stored. Channel image number 3 corresponds to an RGB image. Note that an alpha channel for the image is not included. See the separate tag "image alpha request" for control of alpha channel output.

image channel depth request

Integer number of bits per channel to output. If not given, the bit depth per channel is the one that most closely matches each image to be output and is available to the image file format.

This tag is only used by image write handlers.

Bit depth values are typically between 1 and 32. 8-bit depths are the most common.

image clt request

Integer flag to block or force a color lookup table to be output. If not given, a color lookup table is output, if the image has one and the format can support it. If the tag forces output when a file format cannot support it or blocks output when a format must have it, an error is returned.

This tag is only used by image write handlers. A value of **IMCLTDUMP** forces color lookup table output. A value of **IMCLTNODUMP** blocks output.

image compression request

Integer compression mode to use. If not given, the most common or best compression scheme available to the format is used. If the tag specifies a compression scheme not available to the format, an error is returned.

This tag is only used by image write handlers. The compression scheme selects how pixel data is compressed in order to save space in the output file. Available compression schemes include:

Value	Meaning
IMCOMPNONE	Don’t compress
IMCOMPRLE	Run-length encode
IMCOMPLZW	Lempel-Ziv Welsh encoding
IMCOMPPACKBITS	Apple PackBits

Uncompressed images store each image pixel as a separate value in the file.

Run-length encoded images denote runs of adjacent pixels with the same color. Such runs are reduced to a run length and a run value (the pixel's color) and write these two values to the file instead of the (potentially) much longer run itself. This usually gains about a 30% reduction in storage space. Most image file formats support some form of run-length encoding, but they differ in the exact mechanics.

Lempel-Ziv Welsh and Apple PackBits encoding are bit-wise run-length encoding schemes that are more time-consuming to process but achieve even higher compression ratios. Both schemes are available for only a few image file formats.

image interlace request

Integer RGB image interlace mode to use. If not given, the most common or most easily compressed RGB interlacing scheme is used. If the tag specifies that an interlace mode be used that the image file format cannot support, an error is returned.

This tag is only used by image write handlers, and only if the image to be written is an RGB image. The interlace mode specifies how RGB images should be stored in the file. Available interlace modes include

Value	Meaning
IMINTERNONE	Don't interlace
IMINTERLINE	Scanline interlace
IMINTERPLANE	Plane interlace

Non-interlaced RGB images are stored with each pixel's red, green, and blue components as adjacent data in the file. For instance, an RGB image would be stored as: RGBRGBRGBRGB.

Scanline interlaced images store all of the scanline's red components first, then the scanline's green, then blue components. The next scanline starts over again with just its red components, and so on.

Plane interlaced images store the red components for all of the image pixels, followed by all of the green, then all of the blue.

When compressing images, plane interlacing usually produces the best compression.

image mono threshold

Integer pixel value threshold beyond which grayscale pixel values are considered black, and below which pixel values are considered white when mapping grayscale images to monochrome. If not given, the default is 127.

This tag is only used by image write handlers. The monochrome threshold is used only if the output of the image into the selected file format requires reducing it from color or grayscale to monochrome. Color images are converted to grayscale prior to monochrome thresholding.

See also **ImVfbToMono**(3IM).

image type request

Integer image type to output. If not given, the image type is that of the image being written or one as close to it as supported by the selected output image file format. If this tag is given and the requested type is not available to the format, an error is returned.

This tag is only used by image write handlers. Available image types include

Value	Meaning
IMTYPEINDEX	Color index per pixel
IMTYPERGB	RGB color per pixel

Color index images correspond to **IMVFBINDEX8**, **IMVFBINDEX16**, and **IMVFBMONO** virtual frame buffer field values. RGB images correspond to **IMVFBRGB** virtual frame buffers.

If the selected image type differs from that of an image to be written, the image is converted into a temporary virtual frame buffer before being output.

See also **ImVfbToIndex8**(3IM), **ImVfbToIndex16**(3IM), **ImVfbToRgb**(3IM), **ImVfbToGray**(3IM), and **ImVfbToMono**(3IM).

program name

Character string name of the program, such as the value in *argv[0]* passed into **main**. If not given, the program name defaults to "program."

Program name is used when constructing error message text to be printed to the error stream passed to the error handler (see below).

Error Handling

Upon encountering an error, the image file read and write routines use the following algorithm to decide how to report the error and whether or not to return or try and continue:

```
if an error handler has been defined in the flagsTable,
   if the error is fatal,
      call the error handler with the error message text
      set ImErrNo and return an error
   call the error handler with the error message text
   if the error handler returned a -1,
      set ImErrNo and return an error
else if an error stream has been defined in the flagsTable,
   print the error message text to the stream
   if the error is fatal,
      set ImErrNo and return an error
else if the error is fatal or a warning,
      set ImErrNo and return an error
```

Fatal errors are errors that cannot be recovered from. If a handler has been given, it is called before the read/write routine returns. If a stream has been defined instead, it is printed to before returning. Otherwise, the routine just returns an error code.

Warning errors are errors that somebody should see but don’t require aborting the read or write operation. If a handler has been given, it is called and its return value checked. If it returns a -1, the read/write routine returns. Otherwise it tries to continue. If a stream has been defined instead, the error text is printed without returning. Otherwise, the warning reverts to a fatal error and the routine returns with an error code.

Information errors are messages that should be displayed, but don't require aborting the operation. If a handler has been given, it is called and its return value checked. If it returns -1, the operation is aborted. Otherwise, it is continued. If a stream has been defined instead, the information text is printed out.

Error handlers may be declared as follows:

```
int MyHandler( severity, errno, message )
            int severity;
            int errno;
            char *message;
```

severity can be one of these three error severity types:

severity	**Meaning**
IMERRORFATAL	Fatal error
IMERRORWARNING	Warning error
IMERRORINFO	Information only

The *errno* argument to the handler gives the **ImErrNo** code for the error. See **ImErrNo**(3IM) for information on error numbers and generic error message texts for each.

The *message* argument to the handler is the same as that printed to the error stream, if any, and has the form:

```
program_name: file_name: text
```

The *program_name* and *file_name* are those specified by flags in the *flagsTable* (if given). The *text* of the message describes the error and may or may not be more informative than the generic message available based on the *errno* error code. The error message is not terminated with a carriage return.

The error handler should return -1 if the read or write operation is to be aborted because of the error or 0 if it should be continued. A typical error handler in a windowing-based application would use the *severity* code to decide how to display a dialog box on the screen:

IMERRORFATAL
: Dialog box containing the error message text and an "Abort" button is displayed. Pressing "Abort" takes down the dialog box and returns -1 from the handler.

IMERRORWARNING and IMERRORINFO
: Dialog box containing the error message text and "Abort" and "Continue" buttons are displayed. Pressing "Abort" takes down the dialog box and returns -1 from the handler; pressing "Continue" takes down the dialog box and returns 0 from the handler.

Automatic Conversions

Most of the *flagsTable* flags having names beginning with "image" direct how automatic image conversions should take place when writing an image file. In most cases, the programmer need never use any of these. Automatic conversion takes place and does the "right thing." These flags are only necessary if the programmer wishes to steer automatic conversion a particular way.

Automatic image conversion is necessary when an image to be written out is not in a format acceptable for the image file format. For instance, if an RGB image is to be written out as a MacPaint file, the image must first be converted to monochrome. This is an automatic image conversion.

Image conversions could be done by the user prior to calling **ImFileWrite** or **ImFileFWrite**, but the added hassle makes it convenient for the write routines themselves to handle image conversion.

Image conversion must alter an image's attributes to match those of the output file format. An image has a type (color index or RGB), a number of channels of color data per pixel (usually 1 or 3), and a bit depth for each of those channels (usually 8 bits). The three *flagsTable* flags "image type request," "image channel number request," and "image channel depth request" allow you to constrain automatic conversion to a particular value for these attributes.

If an attribute is not constrained, automatic conversion chooses values that most closely match the image to be written. If those values specify an image that the output image file format cannot support (such as an RGB image in a monochrome MacPaint file), then automatic conversion makes the closest match to what the file format can support. The image to be written is then converted (in a temporary virtual frame buffer) to that match and output.

If an attribute is constrained, automatic conversion restricts its matching algorithm to require the image attribute match the user's request. If this prevents the algorithm from creating a match between an image to be written and an output image file format's supported abilities, then an error is returned and the image is not output.

For instance, an RGB image is to be output into a MacPaint file. With no constraints, automatic conversion will convert the image to 1-bit monochrome and output it. If image type is constrained to be RGB, then automatic conversion cannot make a match (RGB cannot be stored in MacPaint files), and an error is returned.

Automatic conversion contraints are most useful when a particular variant of an output file format is desired. For instance, a Sun rasterfile can support 1-bit monochrome, 8-bit color index, and 24-bit RGB image storage. Without contraints, an incomming RGB image will be written out as an RGB rasterfile (the closest automatic match). If the user wishes to display the image on an 8-bit frame buffer, the "image type request" flag could be used to constrain output conversion to a **IMTYPEINDEX**, color index image. This would force automatic conversion to output an 8-bit color index Sun rasterfile.

Output Constraints

Many image file formats support multiple variants on image compression, RGB image interlacing, and the inclusion of color lookup tables and alpha channels. Many of the *flagsTable* flags having names beginning with "image" direct format output to use a particular variant of the file format.

If not given, these flags default to the "right thing." Image compression defaults to the best or most common compression scheme. RGB image interlace defaults to the most common or most easily compressed interlace mode. Color lookup tables are written if the image has one and the format can store it. The same applies to alpha plane output.

Pipe Handling

When reading from or writing to pipes, some image file formats require that the data be stored in a temporary file. Such a file is created and later destroyed. If there is insufficient space in **/usr/tmp** for this file, an error is returned.

EXAMPLES

The following code opens and reads in a Sun RASterfile and retrieves the new virtual frame buffer image from the *dataTable*. Note that a NULL pointer is given instead of a *flagsTable*.

```
#include <stdio.h>
#include "im.h"

main( )
```

```
{
        FILE *rasFile;
        ImVfb *image;
        TagTable *dataTable;
        TagEntry *imageEntry;

        /* Get space for the data list. */
        dataTable = TagTableAlloc( );

        /* Open the Sun rasterfile. */
        rasFile = fopen( "myfile.ras", "r" );

        /* Read in its images. */
        ImFileFRead( rasFile, "ras", NULL, dataTable );

        /* Get the virtual frame buffer. */
        imageEntry = TagTableQDirect( dataTable, "image vfb" );
        TagEntryQValue( imageEntry, &image );

        /* Do something with it. */
        ...
}
```

The following code accepts command-line arguments to specify an input format name and filename and an output format name and filename. The input file is opened and read in as the selected image file format. The output file is opened and the image data written back out in the new format.

```
#include <stdio.h>
#include "im.h"

main( argc, argv )
        int argc;
        char *argv[ ];
{
        FILE *file;
        TagTable *dataTable;

        /* Get space for the data list. */
        dataTable = TagTableAlloc( );

        /* Open the input file and read it in. */
        file = fopen( argv[2], "r" );
        ImFileFRead( file, argv[1], NULL, dataTable );
        fclose( file );

        /* Open the output file and write it out. */
        file = fopen( argv[4], "w" );
        ImFileFWrite( file, argv[3], NULL, dataTable );
        fclose( file );
}
```

The above program is a complete generic image file converter that can read in any selected image file and output a new one in any desired format. The above code is the basis for the SDSC image conversion tool **imconv**(1IM).

The following code outputs an image stored in a virtual frame buffer and constrains it to be RGB and include an alpha plane.

```
writeImage( image, fileName )
        ImVfb *image;
        char *fileName;
{
        FILE *file;
        TagTable *dataTable;
        TagTable *flagsTable;
        int tmp;

        /* Get space for the data and flags tables. */
        dataTable = TagTableAlloc( );
        flagsTable = TagTableAlloc( );

        /* Add image to data table. */
        TagTableAppend( dataTable,
                TagEntryAlloc( "image vfb", POINTER, &image ) );

        /* Add constraint flags. */
        tmp = IMTYPERGB;
        TagTableAppend( flagsTable,
                TagEntryAlloc( "image type request", INT, &tmp ) );
        tmp = IMALPHADUMP;
        TagTableAppend( flagsTable,
                TagEntryAlloc( "image alpha request", INT, &tmp ) );

        /* Open the output file and write the data. */
        file = fopen( fileName, "w" );
        ImFileFWrite( file, "rlb", flagsTable, dataTable );
        fclose( file );

        /* Clean up. */
        TagTableFree( dataTable );
        TagTableFree( flagsTable );
}
```

RETURNS

On successful completion, the read calls return the number of entries added to the *dataTable* and the write calls return the number of entries used from the *dataTable*.

On failure, the read calls return -1 and set *ImErrNo* to

IMESYS	System call error
IMEMALLOC	Cannot allocate memory
IMEFORMAT	Bad format
IMENOREAD	Read not supported on format
IMEMAGIC	Bad magic number in image file
IMEDEPTH	Unknown image depth

On failure, the read calls return -1 and set *ImErrNo* to

IMESYS	System call error
IMEMANYVFB	Too many VFBs for image write
IMENOVFB	No VFB given for image write
IMEMALLOC	Cannot allocate memory

FILES

/usr/tmp/im.*XXXXXX* *Temporary file for pipe handling*

SEE ALSO

ImIntro (3IM), **ImFileQFormat** (3IM), **TagIntro** (3TAG)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imgif - SDSC CompuServe GIF (Graphics Interchange Format) file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "gif", flagsTable, dataTable )
        int fd;
        TagTable *flagsTable;
        TagTable *dataTable;

int ImFileFRead( fp, "gif", flagsTable, dataTable )
        FILE *fp;
        TagTable *flagsTable;
        TagTable *dataTable;

int ImFileWrite( fd, "gif",  flagsTable, dataTable )
        int fd;
        TagTable *flagsTable;
        TagTable *dataTable;

int ImFileFWrite( fp, "gif",  flagsTable, dataTable )
        FILE *fp;
        TagTable *flagsTable;
        TagTable *dataTable;
```

DESCRIPTION

GIF (Graphics Interchange Format) is CompuServe’s standard for generalized color raster images. This standard is a mechanism to exchange and display high-quality, high-resolution graphics images.

FILE RECOGNITION

CompuServe **gif** files are recognized by the filename suffixes: .gif and .giff.

NOTES

SDSC image library support of CompuServe’s **gif** format contains no proprietary code. CompuServe **gif** format handling is available on any machine for which the SDSC image library is available.

Reading CompuServe GIF files

The SDSC image library reads LZW (Limpel-Ziv Welsh) compressed color index images with depths of 1 through 8 bits, with or without a CLT. 1-bit images are stored as **IMVFBMONO** VFBs, while 2- through 8-bit color index images are stored as **IMVFBINDEX8** VFBs.

If the **gif** file contains multiple images, multiple VFBs are created and appended to the *dataTable*.

Writing CompuServe GIF files

SDSC image library **IMVFBMONO** VFBs are written as LZW compressed 1-bit images, with or without a CLT. **IMVFBINDEX8** VFBs are written as LZW compressed 8-bit images, with or without a CLT.

Other image library VFB types are converted to **IMVFBMONO** or **IMVFBINDEX8** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

The **gif** format can support multiple images in a single file, with the restriction that all such images have the same depth. The SDSC image library currently does not support storage of more than one image in a single **gif** file.

ERRORS

In addition to those listed for **ImFileRead**(3IM), CompuServe **gif** file reading returns the following error codes:

IMEMAGIC	Bad magic number of GIF file
IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), CompuServe **gif** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

GIF Graphics Interchange Format: A standard defining a mechanism for the storage and transmission of raster-based graphics information, CompuServe, June 15, 1987.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Soraya Gonzalez

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

imhdf - SDSC HDF image file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "hdf", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "hdf", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "hdf", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "hdf", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

hdf is a generic, tagged Hierarchical Data File format developed by the National Center for Supercomputing Applications (NCSA). **hdf** files may contain images, scientific data sets, and miscellaneous data items. Such files may be created by several NCSA tools. See the NCSA documentation for details on how to use these tools.

hdf format handling within the SDSC image library is limited to images of certain depths and storage methods.

FILE RECOGNITION

hdf files are recognized by these filename suffixes: .hdf, .df, and .ncsa.

NOTES

SDSC image library support of the **hdf** format does not require use of NCSA's HDF library **libdf.a** and contains no proprietary code. **hdf** is available on any machine for which the SDSC image library is available.

Reading HDF image files

The SDSC image library can read **hdf** image files with one or more images and map them to VFBs as follows:

Depth/color	**Type of compression**	**Mapped to VFB**
8-bit color index	un- and RLE-compressed	IMVFBINDEX8
16-bit color index	uncompressed only	IMVFBINDEX16
24-bit color index	uncompressed only	IMVFBINDEX24
32-bit color index	uncompressed only	IMVFBINDEX32
24-bit RGB	un- and RLE-compressed	IMVFBRGB

All depths except 24-bit RGB may or may not contain a color lookup table (CLT).

RGB images may be non-interlaced, scanline-interlaced, or plane-interlaced.

Note that 24-bit and 32-bit color indexes are truncated to the lower 16 bits when the file is read into an IMVFBINDEX16 VFB.

The HDF IMCOMP compression scheme is not supported. HDF's byte-based RLE compression scheme is only supported on 8-bit color index and 24-bit RGB images.

Administrative header information for **hdf** files is required in MBF (most-significant byte first) byte order by the HDF specification. Images, color tables, and other information in the file may be in either MBF or LBF (least-significant byte first) byte order. *Note:* **hdf** documentation refers to MBF as DFNTI_MBO (Motorola byte order) and LBF as either DFNTI_VBO (VAX byte order) or DFNTI_IBO (Intel byte order).

Floating-point file data may use IEEE, VAX, or CRAY floating-point formats. Character-string file data must be ASCII. EBCDIC is not supported.

If the file's image has a color map, the image library VFB includes a CLT.

If the file contains multiple images, multiple VFBs are appended to the *dataTable*.

Writing HDF image files

Images to be written out are mapped from image library VFBs as follows:

Mapped from VFB	**Depth/color**	**Type of compression**
IMVFBINDEX8	8-bit color index	un- or RLE-compressed
IMVFBINDEX16	16-bit color index	uncompressed only
IMVFBRGB	24-bit RGB	uncompressed only

If the incoming VFB has a CLT, the image written to the **hdf** file contains a CLT.

RGB images may be stored in non-interlaced, scanline-interlaced, and plane-interlaced modes.

Note: Previous versions of this code supported writing of compressed RGB images. This feature has been temporarily removed pending full support for such images by the NCSA and SpyGlass tools. Macintosh-based HDF tools currently do not support 24-bit RGB images. The NCSA UNIX-based HDF tools reliably handle uncompressed RGB images only.

Other image library VFB types are converted to one of the above types prior to being written out. See the **ImFileWrite**(3IM) man page for details.

Administrative header information for **hdf** files is written in MBF byte order as required by the HDF specification. Images, color tables, and other information in the file is written in the same byte order as the host writing the file (MBF or LBF), which ensures the quickest possible I/O for the host. *Note:* **hdf** documentation refers to MBF as DFNTI_MBO (Motorola byte order) and LBF as either DFNTI_VBO (VAX byte order) or DFNTI_IBO (Intel byte order).

Floating-point file data is written using the same floating-point format as the host writing the file (IEEE, VAX, or CRAY). Character-string file data is always ASCII. EBCDIC is not supported.

If the *dataTable* contains multiple images, the output **hdf** file also contains multiple images.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **hdf** file reading returns the following error codes:

IMEMAGIC	Bad Magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Syntax error in HDF file
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), **hdf** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

NCSA HDF, National Center for Supercomputing Applications.

NCSA HDF Calling Interfaces and Utilities, Version 3.1, July 1990, National Center for Supercomputing Applications.

NCSA HDF Specifications, March 1989, National Center for Supercomputing Applications.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imicon - SDSC Sun ICON file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "icon", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFRead( fp, "icon", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileWrite( fd, "icon", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFWrite( fp, "icon", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;
```

DESCRIPTION

icon image files are used by Sun Microsystem's SunView window system, NeWS window system, OpenWindows NeWS tool set, and X11 XView tool set for the storage of icons, cursors, fill patterns, and pieces of widgets (like button check-marks).

Sun **icon** files can be most easily generated using Sun's **iconedit**(1) icon and cursor editor. The Sun operating system release includes a directory of standard icons, cursors, background patterns, and widget pieces in **icon** format in the directory **/usr/include/images**. See the Sun documentation set for details on how to use the tools dealing with Sun **icon** files.

FILE RECOGNITION

Sun **icon** files are recognized by these filename suffixes: .icon, .cursor, and .pr.

NOTES

SDSC image library support of the Sun **icon** format does not require use of Sun libraries and contains no proprietary code. Sun **icon** format handling is available on any machine for which the SDSC image library is available.

icon files contain a C-language array initialization starting with a comment header giving the width and height of the icon, followed by ASCII hex data giving the icon's bitmap. For example,

```
/* Format_version=1, Width=16, Height=16, Depth=1, Valid_bits_per_item=16
 */
     0x0000,0x0000,0x0000,0x0000,0x6200,0x920C,0x1392,0x7250,
     0x9252,0x7B8C,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000
```

Format_version must be 1; there are no other versions at this time.

Width and **Height** are the width and height of the icon, in pixels. Sun's **iconedit**(1) tool requires that the icon width be a multiple of 16. The SDSC image library similarly constrains the icon width for consistency with Sun tools and common usage.

Depth must be 1; there are no other depths in common use.

Valid_bits_per_item must be 16; there are no other sizes in common use.

Icon pixel data is given immediately following the comment header. Monochrome pixels are packed 16 to an integer and written out as a string of 4 hex characters. Image data is given from left to right, top to bottom. Values are comma-separated.

Reading Sun ICON files

The SDSC image library reads Sun **icon** bitmaps and maps them to **IMVFBMONO** VFBs without color lookup tables (CLTs).

Writing Sun ICON files

The SDSC image library writes **IMVFBMONO** VFBs as Sun **icon** bitmaps.

Other image library VFB types are converted to **IMVFBMONO** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

Image width and height fields in the comment header are based on the VFB's width and height. Image width is rounded up to the next multiple of 16 bits. If padding is necessary, pad bits are set to 0 (white).

Warning: The Sun icon editor **iconedit**(1) was designed to edit small bitmaps, like icons and cursors. Consequently, it tends to have severe difficulty with large bitmaps, such as those that can be generated using the SDSC image library.

ERRORS

In addition to those listed for **ImFileRead**(3IM), Sun **icon** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Syntax error in parsing **icon** file
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), Sun **icon** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

iconedit(1) from the Sun OpenWindows **man** page set.

Pixrect Reference Manual, Sun Microsystems.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imiff - SDSC Sun-TAAC file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "iff", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "iff", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "iff", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "iff", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

iff image files are generated by Sun Microsystems TAAC software such as **voxvu**(1) and **cloudvu**(1). See the *TAAC-1 Application Accelerator: Software Reference Manual* for information on how to use these programs.

Note that image files compressed with the Sun-TAAC utility **make_movie**(1) cannot be read by the SDSC image library.

FILE RECOGNITION

iff files are recognized by the following filename suffix: .iff.

NOTES

SDSC image library support of the Sun-TAAC **iff** format does not require use of Sun-TAAC tools or hardware and contains no proprietary code. Sun-TAAC **iff** format handling is available on any machine for which the SDSC image library is available.

Reading Sun-TAAC IFF files

The SDSC image library reads unencoded, non-interlaced, 24-bit RGB and 32-bit RGB+Alpha Sun-TAAC **iff** file images and maps them to **IMVFBRGB** or **(IMVFBRGB | IMVFBALPHA)** VFBs without CLTs.

The library reads unencoded, 8-bit color index Sun-TAAC **iff** images with CLTs and maps them to **IMVFBINDEX8** VFBs with CLTs. Note that color index images never have an alpha plane.

Writing Sun-TAAC IFF files

The SDSC image library writes **IMVFBRGB** and **(IMVFBRGB | IMVFBALPHA)** VFBs to unencoded, non-interlaced 24-bit RGB or 32-bit RGB+Alpha Sun-TAAC **iff** files. Scanline- and plane-interlaced modes are not available in **iff** files.

The image library writes **IMVFBINDEX8** VFBs with CLTs as unencoded, 8-bit color index images with CLTs in Sun-TAAC **iff** files.

Other image library VFB types are converted to **IMVFBRGB** or **IMVFBINDEX8** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

iff does not support any form of image compression.

ERRORS

In addition to those listed for **ImFileRead**(3IM), Sun-TAAC **iff** file reading returns the following error codes:

IMEDEPTH	Unknown image depth
IMEFORMAT	Bad format
IMEMAGIC	Bad magic number
IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in read operation
IMESYNTAX	Syntax error

In addition to those listed for **ImFileWrite**(3IM), Sun-TAAC **iff** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Sun-TAAC User Manual, *Volume Rendering Package,* Sun Microsystems.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

T. Todd Elvins
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

immpnt - SDSC Apple Macintosh MacPaint translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "mpnt", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "mpnt", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "mpnt", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "mpnt", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

The MacPaint **mpnt** file is the standard Apple Macintosh monochrome bitmap image file format. It can be read by many Macintosh graphics applications, and many Macintosh applications that can export bitmap graphics do so in the MacPaint **mpnt** file format.

FILE RECOGNITION

mpnt files are recognized by these filename suffixes: .mpnt, .macp, and .pntg.

NOTES

SDSC image library support of the Apple MacPaint **mpnt** format does not require use of Apple tools or hardware and contains no proprietary code. MacPaint **mpnt** format handling is available on any machine for which the SDSC image library is available.

Reading MacPaint MPNT files

The SDSC image library reads 1-bit MacPaint **mpnt** PackBits compressed images and stores them as **IMVFBMONO** VFBs without a CLT. MacPaint images are always 576 pixels wide by 720 pixels high.

Writing MacPaint MPNT files

SDSC image library **IMVFBMONO** VFBs are written out as 1-bit monochrome MacPaint **mpnt** images. MacPaint images are always compressed using Apples Macintosh PackBits compression scheme.

The MacPaint **mpnt** format requires that images be exactly 576 x 720 pixels. If an image to be written is smaller, the image will be oriented in the upper left corner of the MacPaint image, and the remainder filled with white. If the image to be written is larger, the image will be clipped to keep the upper left corner and warning messages output.

ERRORS

In addition to those listed for **ImFileRead**(3IM), MacPaint **mpnt** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), MacPaint **mpnt** file writing returns the following error codes:

IMEHEIGHT	Image too tall; clipped to 720 pixels high
IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation
IMEWIDTH	Image too wide; clipped to 576 pixels wide

DOCUMENTATION

Inside Macintosh, Volumes I-V, Apple Computer, Inc.

SEE ALSO

imconv (1IM), **imfile** (1IM), **imformats** (1IM), **impict** (3IM)

AUTHOR

John Moreland
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

impbm - SDSC PBM+ suite PBM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "pbm", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "pbm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "pbm", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "pbm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

pbm image files are used by various tools in Jef Poskanzer's PBM+ tool suite. See the PBM documentation set for details on how to use these tools.

FILE RECOGNITION

pbm files are recognized by the filename suffix: .pbm.

NOTES

SDSC image library support of the **pbm** format does not require use of the PBM+ libraries and tools. It does not contain proprietary code or any code from Jef Poskanzer's original PBM+ suite. **pbm** format handling is available on any machine for which the SDSC image library is available.

PBM (Portable Bit Map) started with support for 1-bit monochrome images. Support was added for 8-bit grayscale images (see **impgm**(3IM)), also called PGM (Portable Grayscale Map), and 24-bit RGB images (see **imppm**(3IM)), also called PPM (Portable Pixel Map). Each of these defined a new file format.

The original file formats were ASCII. To reduce the disk space required to store such files, three additional "raw" formats were defined. The raw formats stored the same information as their ASCII counterparts but in binary, reducing the disk space requirement by around 60%.

Today the PBM+ suite contains six related file formats:

PBM	ASCII 1-bit bitmaps
PGM	ASCII 8-bit grayscale pixel maps
PPM	ASCII 24-bit RGB color pixel maps
RPBM	Raw binary 1-bit bitmaps
RPGM	Raw binary 8-bit grayscale pixel maps
RPPM	Raw binary 24-bit RGB color pixel maps

The original PBM suite included several tools. Some handled PBM files but not PGM or PPM. Others handled PGM files, but not PBM or PPM. Then the newer PBM+ distribution introduced a set of tools that recognized all six file formats. These tools generically referred to image files as PNM (Portable aNy Map) files.

The SDSC image library treats the six PBM+ file formats separately. This **man** page only discusses the PBM file format (ASCII monochrome). The remaining file formats are disucussed in their own **man** pages.

Reading PBM image files

For compatibility with the PBM+ suite, the SDSC image library can read any of the PBM+ file formats when the **pbm** format name is used. PBM+ files are read in and mapped to VFBs as follows:

File format	**Depth/color**	**Mapped to VFB**
PBM	1-bit monochrome	**IMVFBMONO** without a CLT
PGM	8-bit color index	**IMVFBINDEX8** without a CLT
PPM	24-bit RGB	**IMVFBRGB** without a CLT
RPBM	1-bit monochrome	**IMVFBMONO** without a CLT
RPGM	8-bit color index	**IMVFBINDEX8** without a CLT
RPPM	24-bit RGB	**IMVFBRGB** without a CLT

White space and comments (lines starting with # and extending to the end of the line) are ignored. White space and comments are not allowed within the raw binary image body.

Writing PBM image files

The SDSC image library writes **IMVFBMONO** VFBs as **pbm** monochrome ASCII bitmap files.

pbm files support no compression schemes.

Other image library VFB types are converted to **IMVFBMONO** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **pbm** file reading returns the following error codes:

IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Premature EOF
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), **pbm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

pbm(5), **pgm**(5), and **ppm**(5) from the PBM+ **man** page set.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **impgm**(3IM), **impnm**(3IM), **imppm**(3IM), **imrpbm**(3IM), **imrpgm**(3IM), **imrpnm**(3IM), **imrppm**(3IM)

AUTHORS

Dave Nadeau and Don Doering
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

impcx - SDSC ZSoft IBM PC Paintbrush file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "pcx", flagsTable, dataTable )
        int       fd;
        TagTable  *flagsTable;
        TagTable  *dataTable;

int ImFileFRead( fp, "pcx", flagsTable, dataTable )
        FILE      *fp;
        TagTable  *flagsTable;
        TagTable  *dataTable;

int ImFileWrite( fd, "pcx", flagsTable, dataTable )
        int       fd;
        TagTable  *flagsTable;
        TagTable  *dataTable;

int ImFileFWrite( fp, "pcx", flagsTable, dataTable )
        FILE      *fp;
        TagTable  *flagsTable;
        TagTable  *dataTable;
```

DESCRIPTION

The **pcx** image file format was invented by ZSoft for use in its IBM PC **PC Paintbrush** tool series. It has become a defacto standard in the IBM PC world and is regularly used for the storage of monochrome and color pixel information by paint-type tools.

See the documentation for each of the IBM PC tools for details on how to use them.

FILE RECOGNITION

ZSoft **pcx** files are recognized by these filename suffixes: .pcx and .pcc.

NOTES

SDSC image library support of the ZSoft **pcx** format does not require use of IBM PC graphics or compute hardware and contains no proprietary code. ZSoft **pcx** format handling is available on any machine for which the SDSC image library is available.

Reading ZSoft PCX files

The SDSC image library can read ZSoft **pcx** files and map them to VFBs are as follows:

Depth/color	**Mapped to VFB**
1-bit monochrome	IMVFBMONO without CLT
2-bit color index	IMVFBINDEX8 with or without CLT

3-bit color index	IMVFBINDEX8 with or without CLT
4-bit color index	IMVFBINDEX8 with or without CLT
8-bit color index	IMVFBINDEX8 with or without CLT

2-, 3-, 4- and 8-bit depths may or may not contain a pallete (color lookup table).

Writing ZSoft PCX files

Images to be written out are mapped from image library VFBs are as follows:

Mapped from VFB	**Depth/color**
IMVFBMONO	1-bit monochrome
IMVFBINDEX8	8-bit color index

If the incoming VFB has a CLT, the image written to the **pcx** file contains a pallete.

pcx files are always compressed with a variant of RLE encoding. However, ZSoft's **pcx** RLE encoding scheme was poorly designed. In the worst case, a compressed **pcx** file may require *double* the disk space that an uncompressed image would occupy. Unfortunately, ZSoft's **pcx** file format does not allow for uncompressed images or compression schemes other than their faulty RLE scheme.

Other image library VFB types are converted to one of the above types prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), ZSoft **pcx** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Unknown parameter in file header
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), ZSoft **pcx** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Technical Reference Manual, ZSoft Corporation, 1988.

Bit-Mapped Graphics, Steve Rimmer, Windcrest/McGraw-Hill publishing, 1990.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

ImPError - print an error message from a virtual frame buffer routine
ImQError - query the error message for the current error number
ImErrNo - give the error number that came from a virtual frame buffer routine

SYNOPSIS

#include "im.h"

extern int ImErrNo ;
extern int ImNErr ;
extern char ∗ImErrList[ ] ;

void ImPError(str)
char ∗str ;

char ∗ImQError( )

DESCRIPTION

ImPError produces a short message on **stderr** describing the last error encountered during a call to an IM package procedure. The argument string *str* is printed first, then a colon and a blank, then the message and a new line. To be of most use, the argument string should include the name of the program or routine that incurred the error. The error number is taken from the external variable **ImErrNo**, which is set when IM errors occur, but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings **ImErrList** is provided; **ImErrNo** can be used as an index into this table to get the message string without the newline. **ImNErr** is the number of messages provided for in the table; it should be checked because new error codes might be added to the system before they are added to the table.

ImQError may be used to query the error message list and return a pointer to the error text associated with the error value of **ImErrNo**.

EXAMPLE

ImPError should be called when an error condition is detected upon return from an IM routine. For example:

```
ReadFb()
{
      ImVfb ∗srcVfb ;
 ...
      srcVfb = ImVfbAlloc( 1280, 1024, IMVFBRGB ) ;
      if( srcVfb == IMVFBNULL )
      {
            ImPError( "ReadFb" ) ;
            exit( 1 ) ;
      }
 ...
}
```

RETURNS

ImPError returns nothing.

ImQError returns the character string message for the current value of **ImErrNo**, or "Unknown error" if there is no message for the error number.

NOTES

If **ImErrNo** is **IMESYS**, indicating a system call error occurred, **ImPError** calls **perror**(3) to print the system call's error message instead of **IMESYS**, and **ImQError** returns the error text associated with **errno**.

SEE ALSO

imintro (3IM), **errno** (2), **perror** (3)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

impgm - SDSC PBM+ suite PGM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "pgm", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "pgm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "pgm", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "pgm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

pgm image files are used by various tools in Jef Poskanzer's PBM+ tool suite. See the PBM documentation set for details on how to use these tools.

FILE RECOGNITION

pgm files are recognized by the filename suffix: .pgm.

NOTES

SDSC image library support of the **pgm** format does not require use of the PBM+ libraries and tools. It does not contain proprietary code or any code from Jef Poskanzer's original PBM+ suite. **pgm** format handling is available on any machine for which the SDSC image library is available.

PBM (Portable Bit Map) started with support for 1-bit monochrome images (see **impbm**(3IM)). Support was added for 8-bit grayscale images, also called PGM (Portable Grayscale Map), and 24-bit RGB images (see **imppm**(3IM)), also called PPM (Portable Pixel Map). Each of these defined a new file format.

The original file formats were ASCII. To reduce the disk space required to store such files, three additional "raw" formats were defined. The raw formats stored the same information as their ASCII counterparts but in binary, reducing the disk space requirement by around 60%.

Today the PBM+ suite contains six related file formats:

PBM	ASCII 1-bit bitmaps
PGM	ASCII 8-bit grayscale pixel maps
PPM	ASCII 24-bit RGB color pixel maps
RPBM	Raw binary 1-bit bitmaps
RPGM	Raw binary 8-bit grayscale pixel maps
RPPM	Raw binary 24-bit RGB color pixel maps

The original PBM suite included several tools. Some handled PBM files but not PGM or PPM. Others handled PGM files but not PBM or PPM. Then the newer PBM+ distribution introduced a set of tools that recognized all six file formats. These tools generically referred to image files as PNM (Portable aNy Map) files.

The SDSC image library treats the six PBM+ file formats separately. This **man** page discusses only the PGM file format (ASCII grayscale). The remaining file formats are disucussed in their own **man** pages.

Reading PGM image files

For compatibility with the PBM+ suite, the SDSC image library can read any of the PBM+ file formats when the **pgm** format name is used. PBM+ files are read in and mapped to VFBs as follows:

File format	**Depth/color**	**Mapped to VFB**
PBM	1-bit monochrome	**IMVFBMONO** without a CLT
PGM	8-bit color index	**IMVFBINDEX8** without a CLT
PPM	24-bit RGB	**IMVFBRGB** without a CLT
RPBM	1-bit monochrome	**IMVFBMONO** without a CLT
RPGM	8-bit color index	**IMVFBINDEX8** without a CLT
RPPM	24-bit RGB	**IMVFBRGB** without a CLT

White space and comments (lines starting with # and extending to the end of the line) are ignored. White space and comments are not allowed within the raw binary image body.

Writing PGM image files

The SDSC image library writes **IMVFBINDEX8** VFBs without CLTs as **pgm** grayscale bitmap files.

pgm files support no compression schemes.

Other image library VFB types are converted to **IMVFBINDEX8** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **pgm** file reading returns the following error codes:

IMEMAGIC	Bad Magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Premature EOF
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), **pgm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

pbm(5), **pgm**(5), and **ppm**(5) from the PBM+ **man** page set.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **impbm**(3IM), **impnm**(3IM), **imppm**(3IM), **imrpbm**(3IM), **imrpgm**(3IM), **imrpnm**(3IM), **imrppm**(3IM)

AUTHORS

Dave Nadeau and Don Doering
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

impic - SDSC PIXAR PIC file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "pic", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFRead( fp, "pic", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileWrite( fd, "pic", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFWrite( fp, "pic", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;
```

DESCRIPTION

pic image files are generated by PIXAR programming software, such as **ChapVolumes** and **ChapReyes**, the PIXAR Image Runtime Library called **Pirl**, and the PIXAR rendering tool **RenderMan**. See the PIXAR documentation set for details on how to use these applications and tools.

PIXAR's **pic** file format is sometimes referred to as PICIO in PIXAR documentation. **pic** and PICIO mean the same thing.

PIXAR's **xpic** is not the same as **pic** and is *not* supported by the SDSC image library.

Note: PIXAR's **RenderMan** always saves its image files with .pic filename suffixes. However, depending upon output defaults, **RenderMan** can generate .pic files with PICIO (same as **pic**) data or TIFF data. .pic files with TIFF data will confuse the SDSC image library. For you to avoid generating this type of file, we recommend that you configure **RenderMan** defaults to generate .pic files with PICIO data.

FILE RECOGNITION

PIXAR **pic** files are recognized by the following filename suffixes: .pic, .picio, and .pixar.

NOTES

SDSC image library support of the PIXAR **pic** format does not require use of PIXAR's tools, libraries, or hardware and contains no proprietary code. PIXAR **pic** format handling is available on any machine for which the SDSC image library is available.

Reading PIXAR image files

The SDSC image library can read PIXAR **pic** files and map them to VFBs as follows:

Depth/color	**Type of compression**	**Mapped to VFB**
8-bit R	Dump and encoded	IMVFBINDEX8 without CLT
12-bit R	Dump and encoded	IMVFBINDEX16 without CLT
24-bit RGB	Dump and encoded	IMVFBRGB without CLT
36-bit RGB	Dump and encoded	IMVFBRGB without CLT
32-bit RGB-Alpha	Dump and encoded	(IMVFBRGB | IMVFBALPHA) without CLT
48-bit RGB-Alpha	Dump and encoded	(IMVFBRGB | IMVFBALPHA) without CLT

Single-channel images in 8- and 12-bit depths store the channel's data as the red image channel. The **pic** code treats such images as grayscale images without color lookup tables (CLTs).

Dump format images are stored uncompressed; encoded format images are stored using a variant of run-length-encoded (RLE) compression.

A PIXAR **pic** image can be represented in files as a series of tiles, each one containing a part of the image. Such image tiling is *not* supported by the SDSC image library.

Note: RGB images with 12-bit data channels (i.e., 36-bit RGB and 48-bit RGB-Alpha) are reduced to 8-bit data channels when read in.

Writing PIXAR image files

SDSC image library VFBs are written to **pic** files as follows:

Mapped from VFB	**Depth/color**	**Type of compression**
IMVFBINDEX8	8-bit R	dump and encoded
IMVFBINDEX16	8-bit R	dump and encoded
IMVFBRGB	24-bit RGB	dump and encoded
(IMVFBRGB | IMVFBALPHA)	32-bit RGB-Alpha	dump and encoded

None of the output formats supports storing a CLT with the image in the file.

Other image library VFB types, or VFBs with CLTs, are converted to one of the above types prior to being written out. See the **ImFileWrite**(3IM) man page for details.

In SDSC image library terminology, dump files are uncompressed; encoded files use a variant of run-length-encoded (RLE) compression.

RGB and RGB-Alpha images are always stored non-interlaced (i.e., RGBRGBRGB...). Scanline- and plane-interlaced modes are not supported.

ERRORS

In addition to those listed for **ImFileRead**(3IM), PIXAR **pic** file reading returns the following error codes:

IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Unknown image type, mode, or multiple tiles
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**, PIXAR **pic** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

PIXAR Image Computer Programmer's Manual, PIXAR.

PIXAR Image Computer ChapLibraries User's Guide, PIXAR.

The RenderMan Companion, Steve Upstill, PIXAR.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Jim McLeod
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

impict - SDSC Apple Macintosh PICT translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "pict", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "pict", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "pict", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "pict", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

The **PICT** file is the standard Apple Macintosh image file format. It can be read by almost any Macintosh graphics application. Most Macintosh applications that can export graphics do so in the **PICT** file format.

FILE RECOGNITION

pict files are recognized by these filename suffixes: .pict and .pict2.

NOTES

SDSC image library support of the Apple **pict** format does not require use of Apple tools or hardware and contains no proprietary code. Apple **pict** format handling is available on any machine for which the SDSC image library is available.

Reading Apple PICT files

The SDSC image library reads 1-, 2-, 4-, 8-, and 16-bit color index images, with or without a CLT, compressed using Apple's PackBits, and maps them into **IMVFBRGB** VFBs.

Note that all **pict** images are read into **IMVFBRGB** VFBs. This is necessary because of a **pict** feature that allows each piece of an image to have its own CLT. This can easily exceed any reasonable size color index image CLT very quickly. To avoid this, incoming image pieces in a **pict** file are always converted and stored into an RGB VFB.

Both type 1 (monochrome and "old" color) and type 2 (full color) **pict** files are handled.

Writing Apple PICT files

SDSC image library **IMVFBINDEX8** VFBs are written out to Apple **pict** files as 8-bit color index images with a CLT and compressed using Apple's PackBits compression. Written images are always type 2 **pict** files.

ERRORS

In addition to those listed for **ImFileRead**(3IM), Apple **pict** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Syntax error
IMESYS	System call error in read operation
IMEUNSUPPORTED	Unsupported opcode

In addition to those listed for **ImFileWrite**(3IM), Apple **pict** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Inside Macintosh, Volumes I-V, Apple Computer, Inc.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **immpnt**(3IM)

AUTHOR

John Moreland
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

impix - SDSC Alias PIX file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "pix", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "pix", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "pix", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "pix", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

pix image files are generated by the rendering and painting tools of Alias Research, Inc., such as **renderer**, **raytracer**, and **paint**. See the Alias documentation set for details on how to use these tools.

Note: Alias **quickpaint**, available on Silicon Graphics, Inc., IRIS workstations, uses Silicon Graphic’s **rgb** image file format rather than the Alias **pix** image file format. See the **imrgb**(3IM) **man** page for details on the **rgb** format.

FILE RECOGNITION

Alias **pix** files are recognized by these following filename suffixes: .alias and .pix.

NOTES

SDSC image library support of the Alias **pix** format does not require use of Alias tools and contains no proprietary code. Alias **pix** format handling is available on any machine for which the SDSC image library is available.

Reading Alias PIX files

The SDSC image library reads run-length-encoded (RLE), noninterlaced, 24-bit RGB Alias **pix** files and maps them to **IMVFBRGB** VFBs without color lookup tables (CLTs).

Writing Alias PIX files

Image library **IMVFBRGB** VFBs are written to run-length-encoded, noninterlaced, 24-bit RGB Alias **pix** files. Scanline- and plane-interlaced modes are not supported by Alias **pix** files.

Other image library VFB types are converted to **IMVFBRGB** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), Alias **pix** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), Alias **pix** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Alias Reference Manual, Alias Research, Inc.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

impnm - SDSC Jef Poskanzer's PBM+ suite PNM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "pnm", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "pnm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "pnm", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "pnm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

pnm image files are used by various tools in Jef Poskanzer's PBM+ tool suite. See the PBM documentation set for details on how to use these tools.

FILE RECOGNITION

pnm files are recognized by the filename suffix: .pnm.

NOTES

SDSC image library support of the **pnm** format does not require use of the PBM+ libraries and tools. It does not contain proprietary code or any code from Jef Poskanzer's original PBM+ suite. **pnm** format handling is available on any machine for which the SDSC image library is available.

PBM (Portable Bit Map) started with support for 1-bit monochrome images (see **impbm**(3IM)). Support was added for 8-bit grayscale images (see **impgm**(3IM)), also called PGM (Portable Grayscale Map), and 24-bit RGB images (see **imppm**(3IM)), also called PPM (Portable Pixel Map). Each of these defined a new file format.

The original file formats were ASCII. To reduce the disk space required to store such files, three additional "raw" formats were defined. The raw formats stored the same information as their ASCII counterparts but in binary, reducing the disk space requirement by around 60%.

Today the PBM+ suite contains six related file formats:

PBM	ASCII 1-bit bitmaps
PGM	ASCII 8-bit grayscale pixel maps
PPM	ASCII 24-bit RGB color pixel maps
RPBM	Raw binary 1-bit bitmaps
RPGM	Raw binary 8-bit grayscale pixel maps
RPPM	Raw binary 24-bit RGB color pixel maps

The original PBM suite included several tools. Some handled PBM files but not PGM or PPM. Others handled PGM files but not PBM or PPM. Then the newer PBM+ distribution introduced a set of tools that recognized all six file formats. These tools generically referred to image files as PNM (Portable aNy Map) files.

The SDSC image library treats the six PBM+ file formats separately. However, in order to be compatible with the apparent trend in the PBM+ tool set, the new generic **pnm** and **rpnm** names mean "any of the PBM+ formats." This **man** page only discusses the PNM generic file name. The remaining file formats are disucussed in their own **man** pages.

Reading PNM image files

The SDSC image library can read any of the PBM+ file formats when the **pnm** format name is used. This includes the raw binary variants as well as the ASCII format variants. PBM+ files are read in and mapped to VFBs as follows:

File format	Depth/color	Mapped to VFB
PBM	1-bit monochrome	**IMVFBMONO** without a CLT
PGM	8-bit color index	**IMVFBINDEX8** without a CLT
PPM	24-bit RGB	**IMVFBRGB** without a CLT
RPBM	1-bit monochrome	**IMVFBMONO** without a CLT
RPGM	8-bit color index	**IMVFBINDEX8** without a CLT
RPPM	24-bit RGB	**IMVFBRGB** without a CLT

Header white space and comments (lines starting with # and extending to the end of the line) are ignored. White space and comments are not allowed within the raw binary image body.

Writing PNM image files

The SDSC image library can write PBM+ suite files in the following configurations:

Mapped from VFB	Depth/color	File format
IMVFBMONO without a CLT	1-bit monochrome	PBM
IMVFBINDEX8 without a CLT	8-bit color index	PGM
IMVFBRGB without a CLT	24-bit RGB	PPM

RGB images are always stored noninterlaced (i.e., RGBRGBRGB...). Scanline- and plane-interlaced modes are not available in PPM.

All PBM+ suite files support no compression schemes.

Other image library VFB types are converted to one of the above types prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **pnm** file reading returns the following error codes:

IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Premature EOF
IMESYS	System call error in read operation

In addition to those for **ImFileWrite**(3IM), **pnm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

pbm(5), **pgm**(5), and **ppm**(5) from the PBM+ **man** page set.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **impbm**(3IM), **impgm**(3IM), **imppm**(3IM), **imrpbm**(3IM), **imrpgm**(3IM), **imrpnm**(3IM), **imrppm**(3IM)

AUTHORS

Dave Nadeau and Don Doering
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imppm - SDSC PBM+ suite PPM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "ppm", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFRead( fp, "ppm", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileWrite( fd, "ppm", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFWrite( fp, "ppm", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;
```

DESCRIPTION

ppm image files are used by various tools in Jef Poskanzer's PBM+ tool suite. See the PBM documentation set for details on how to use these tools.

FILE RECOGNITION

ppm files are recognized by the filename suffix: .ppm.

NOTES

SDSC image library support of the **ppm** format does not require use of the PBM+ libraries and tools. It does not contain proprietary code or any code from Jef Poskanzer's original PBM+ suite. **ppm** format handling is available on any machine for which the SDSC image library is available.

PBM (Portable Bit Map) started with support for 1-bit monochrome images (see **impbm**(3IM)). Support was added for 8-bit grayscale images (see **impgm**(3IM)), also called PGM (Portable Grayscale Map), and 24-bit RGB images, also called PPM (Portable Pixel Map). Each of these defined a new file format.

The original file formats were ASCII. To reduce the disk space required to store such files, three additional "raw" formats were defined. The raw formats stored the same information as their ASCII counterparts but in binary, reducing the disk space requirement by around 60%.

Today the PBM+ suite contains six related file formats:

PBM	ASCII 1-bit bitmaps
PGM	ASCII 8-bit grayscale pixel maps
PPM	ASCII 24-bit RGB color pixel maps
RPBM	Raw binary 1-bit bitmaps
RPGM	Raw binary 8-bit grayscale pixel maps
RPPM	Raw binary 24-bit RGB color pixel maps

The original PBM suite included a variety of tools. Some handled PBM files, but not PGM or PPM. Others handled PGM files, but not PBM or PPM. Then the newer PBM+ distribution introduced a set of tools that recognized all six file formats. These tools generically referred to image files as PNM files, which stands for "Portable aNy Map."

The SDSC image library treats the six PBM+ file formats separately. This **man** page only discusses the PPM file format (ASCII RGB). The remaining file formats are discussed in their own **man** pages.

Reading PPM image files

For compatibility with the PBM+ suite, the SDSC image library can read any of the PBM+ file formats when the **ppm** format name is used. PBM+ files are read in and mapped to VFBs as follows:

File format	**Depth/color**	**Mapped to VFB**
PBM	1-bit monochrome	**IMVFBMONO** without a CLT
PGM	8-bit color index	**IMVFBINDEX8** without a CLT
PPM	24-bit RGB	**IMVFBRGB** without a CLT
RPBM	1-bit monochrome	**IMVFBMONO** without a CLT
RPGM	8-bit color index	**IMVFBINDEX8** without a CLT
RPPM	24-bit RGB	**IMVFBRGB** without a CLT

White space, and comments starting with # and extending to the end of the line are ignored. White space and comments are not allowed within the raw binary image body.

Writing PPM image files

The SDSC image library writes **IMVFBRGB** VFBs as **ppm** RGB files.

ppm images are always stored noninterlaced (i.e., RGBRGBRGB...). Scanline- and plane-interlaced modes are not available in **ppm** files.

ppm files support no compression schemes.

Other image library VFB types are converted to **IMVFBRGB** VFBs prior to being written out. See the **ImFileWrite**(3IM) man page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **ppm** file reading returns the following error codes:

IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Premature EOF
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), **ppm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

pbm(5), **pgm**(5), and **ppm**(5) from the PBM+ **man** page set.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **impbm**(3IM), **impgm**(3IM), **impnm**(3IM), **imrpbm**(3IM), **imrpgm**(3IM), **imrpnm**(3IM), **imrppm**(3IM)

AUTHORS

Dave Nadeau and Don Doering
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imps - SDSC PostScript file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileWrite( fd, "ps", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFWrite( fp, "ps", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;
```

DESCRIPTION

PostScript color, grayscale, and black-and-white image files are used by a variety of PostScript laser printers and windowing system tools.

FILE RECOGNITION

PostScript files are recognized by the following filename suffixes: .ps and .postscript.

NOTES

SDSC image library support of PostScript does not require use of any windowing system libraries or PostScript interpreters, and contains no proprietary code. PostScript format handling is available on any machine for which the SDSC image library is available.

PostScript is a reverse polish notation-style, FORTH-like programming language used to describe text and pictures to be rendered by a PostScript interpreter embedded within a laser printer (such as Apple's LaserWriter printers), windowing system (such as Sun's NeWS or NeXT's NeXTstep), or other display tool. The language contains numerous basic programming language constructs and lots of drawing operations.

PostScript was designed for ease of parsing, not ease of programming. As a result, most people use PostScript as a "blackbox" file format that describes their picture. It is left up to software, such as the SDSC image library, to figure out how to deal with PostScript.

Reading PostScript files

Reading of PostScript is not supported by the SDSC image library. Inclusion of a full PostScript language interpreter in order to read PostScript files would require a great deal of code. This approach is not really practical.

Users needing to read in PostScript data should investigate PostScript interpreters, such as Sun Microsystem's NeWS window system and its image viewer **PageView**(1), or GNU's **GostScript** PostScript-clone interpreter.

Writing PostScript files

The SDSC image library writes color, grayscale, or monochrome VFBs as PostScript. In each case the generated PostScript consists of a header followed by the image data as ASCII hex numbers. The choice of header and the format of the hex numbers depends upon the type of image being written.

IMVFBMONO VFBs are written with a PostScript header that displays the image in black-and-white on black-and-white or color devices. Image data is written as two hex characters for each group of eight adjacent monochrome pixels (as required by the PostScript image operator).

IMVFBINDEX8 VFBs without color lookup tables are written with a PostScript header that displays the image in shades of gray on black-and-white or color devices. Image data is written as two hex characters for each grayscale pixel (as required by the PostScript image operator).

IMVFBINDEX8 VFBs with a color lookup table and **IMVFBRGB** VFBs are written with a PostScript header that displays the image in color on color devices and in shades of gray on black-and-white devices. The PostScript header code checks the device's systemdict for support of the colorimage operator. If the operator exists, the device supports color. In both cases, image data is written as six hex characters at two characters each for the red, green, and blue color component of each pixel (as required by the PostScript colorimage operator). On color devices, this color pixel data generates a color image. On monochrome devices, the header code automatically converts the color pixel data to shades of gray as it is being displayed or printed. This allows the same color PostScript file to be sent to color or black-and-white printers without any change to the file.

Other image library VFB types are converted to one of the above prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

Image data is always generated for pixels in left to right order, from the top of the image to the bottom.

In all cases the image is rotated and scaled up automatically to fill a maximum page area without distorting the image. You can override this automatic orientation and sizing by editing the generated PostScript file. The header of the file contains comments describing how to do this.

PostScript generated by the SDSC image library conforms to Adobe's version 3.0 document structuring conventions.

Warning: All PostScript laser printers have a fixed amount of memory in which to buffer incoming data and build up the image prior to printing. Large images converted to PostScript and then sent to a laser printer may exceed the buffering and image-building memory limitations of the printer. Results vary from printer to printer: some lock up and require a reset, while others silently ignore the images and do not print anything.

ERRORS

In addition to those listed for **ImFileWrite**(3IM), PostScript file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

PostScript Language - Tutorial and Cookbook, Adobe Systems Incorporated.

PostScript Language - Reference Manual, Second Edition, Adobe Systems Incorporated.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHORS

Dave Nadeau
San Diego Supercomputer Center

Loren "Buck" Buchanan
Naval Research Laboratory
Kestrel Associates, Inc.

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imras - SDSC Sun Rasterfile translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "ras", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFRead( fp, "ras", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileWrite( fd, "ras", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFWrite( fp, "ras", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;
```

DESCRIPTION

ras image files are used by various Sun Microsystems Inc. tools, such as **screendump**(1) and **screenload**(1). See the Sun documentation set for details on how to use these tools.

FILE RECOGNITION

Sun **ras** files are recognized by any of the following filename suffixes: .ras, .scr, .sr, and .sun.

NOTES

SDSC image library support of the Sun **ras** format does not require use of Sun's pixrect library or hardware, and contains no proprietary code. Sun **ras** format handling is available on any machine for which the SDSC image library is available.

Reading Sun Rasterfiles

The SDSC image library can read Sun **ras** files and map them to VFBs as follows:

Depth/color	Type of compression	Mapped to VFB
1-bit monochrome	RT_OLD, RT_STANDARD, or RT_BYTE_ENCODED	IMVFBMONO
8-bit color index	RT_OLD, RT_STANDARD,	IMVFBINDEX8

	or RT_BYTE_ENCODED	
24-bit RGB	RT_OLD, RT_STANDARD, or RT_BYTE_ENCODED	IMVFBRGB
32-bit RGB-Alpha	RT_OLD, RT_STANDARD, or RT_BYTE_ENCODED	(IMVFBRGB | IMVFBALPHA)

All depths may or may not contain a color map (color lookup table, or CLT), though, normally, only 8-bit images include one.

In SDSC image library terminology, **RT_STANDARD** and **RT_OLD** files are uncompressed; **RT_BYTE_ENCODED** files use runlength-encoded RLE compression.

If the file's image has a color map, the image library VFB includes a CLT.

Writing Sun Rasterfiles

Images to be written out are mapped from image library VFBs as follows:

Mapped from VFB	**Depth/color**	**Types of compression**
IMVFBMONO	1-bit monochrome	RT_STANDARD, RT_BYTE_ENCODED
IMVFBINDEX8	8-bit color index	RT_STANDARD, RT_BYTE_ENCODED
IMVFBRGB	24-bit RGB	RT_STANDARD, RT_BYTE_ENCODED
(IMVFBRGB | IMVFBALPHA)	32-bit RGB-Alpha	RT_STANDARD, RT_BYTE_ENCODED

If the incoming VFB has a CLT, the image written to the raster file contains a color map.

In SDSC image library terminology, **RT_STANDARD** and **RT_OLD** files are uncompressed; **RT_BYTE_ENCODED** files use runlength-encoded RLE compression. **RT_OLD** raster files cannot be generated.

RGB and RGB-Alpha images are always stored noninterlaced (i.e., RGBRGBRGB...). Scanline- and plane-interlaced modes are not available in the **ras** format.

Other image library VFB types are converted to one of the above types prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), Sun **ras** file reading returns the following error codes:

IMECLTLENGTH	CLT length in file header is strange
IMEDEPTH	Unknown image depth in file header
IMEIMAGETYPE	Unknown image type in file header
IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Unknown CLT type in file header
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), Sun **ras** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

See the comments in **/usr/include/rasterfile.h** on Sun systems.

Pixrect Reference Manual, Sun Microsystems.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imrgb - SDSC Silicon Graphics RGB file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "rgb", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "rgb", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "rgb", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "rgb", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

rgb image files are generated by Silicon Graphics, Inc., software such as **icut**(1) and **snapshot**(1). See the Silicon Graphics documentation for information on how to use these and other Silicon Graphics programs.

FILE RECOGNITION

Silicon Graphics **rgb** files are recognized by the following filename suffixes: .rgb, .iris, and .sgi.

NOTES

SDSC image library support of the Silicon Graphics **rgb** format does not require use of any Silicon Graphics library or hardware and contains no proprietary code. Silicon Graphics **rgb** format handling is available on any machine for which the SDSC image library is available.

Reading Silicon Graphics RGB files

The SDSC image library reads uncompressed and runlength-encoded (RLE), scanline-interlaced, 24-bit RGB Silicon Graphics **rgb** images and stores them as **IMVFBRGB** VFBs without CLTs.

rgb greyscale images are not supported.

Writing Silicon Graphics RGB files

Image library **IMVFBRGB** VFBs are written to uncompressed or runlength-encoded, scanline-interlaced, 24-bit RGB Silicon Graphics **rgb** files. Noninterlaced and plane-interlaced modes are not available in the Silicon Graphics **rgb** format.

Other image library VFB types are converted to **IMVFBRGB** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

rgb greyscale images are not supported.

ERRORS

In addition to those listed for **ImFileRead**(3IM), Silicon Graphics **rgb** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), Silicon Graphics **rgb** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Silicon Graphics RGB Specification, Silicon Graphics Inc.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Jesus Ferrer and T. Todd Elvins
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

imrla - SDSC Wavefront RLA and RLB file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "rla", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "rla", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "rla", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "rla", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

rla image files are generated by Wavefront Technologies, Inc., software such as **image** and **tdv**. See the Wavefront documentation for information on how to use these and other Wavefront programs.

rla is a subset of the newer Wavefront **rlb** specification. Programs that can read and write **rlb** files can also read and write **rla** files. The SDSC image library reader/writer is written to accommodate both types of files.

FILE RECOGNITION

Wavefront **rla** files are recognized by the following filename suffixes: .rla and .rlb.

NOTES

SDSC image library support of the Wavefront **rla** format does not require use of any Wavefront tools and contains no proprietary code. Wavefront **rla** is available on any machine for which the SDSC image library is available.

Reading Wavefront RLA files

The SDSC image library reads runlength-encoded (RLE), scanline-interlaced, 24-bit RGB and 32-bit RGB-Alpha Wavefront files and maps them to **IMVFBRGB** and **(IMVFBRGB | IMVFBALPHA)** VFBs without color lookup tables (CLTs).

Writing Wavefront RLA files

Image library **IMVFBRGB** VFBs with or without alpha planes are written to runlength-encoded, scanline-interlaced, 32-bit RGB-Alpha Wavefront **rla** files. If the incoming VFB does not have an alpha plane, outgoing image file alpha values are all set to 255. Noninterlaced and plane-interlaced modes are not supported by Wavefront **rla** files.

Other image library VFB types are converted to **(IMVFBRGB | IMVFBALPHA)** prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

Note that the Wavefront **rla** specification requires the presence of an alpha channel in an **rla** image file, yet at least one of the Wavefront programs does not include alpha channels in the images that it writes to files.

ERRORS

In addition to those listed for **ImFileRead**(3IM), Wavefront **rla** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Syntax error
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), Wavefront **rla** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Wavefront RLA Specification, Wavefront Technologies, Inc.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

T. Todd Elvins
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

imrle - SDSC Utah RLE file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "rle", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFRead( fp, "rle", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileWrite( fd, "rle", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFWrite( fp, "rle", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;
```

DESCRIPTION

rle image files are generated by the tools of Utah’s Raster Toolkit. See the Utah documentation set for details on how to use these tools.

FILE RECOGNITION

Utah **rle** files are only recognized by the following filename suffix: .rle

NOTES

SDSC image library support of the Utah **rle** format does not require use of any Utah tools and contains no proprietary code. Utah **rle** format handling is available on any machine for which the SDSC image library is available.

The Utah **rle** format is very flexible and allows the definition of a wide range of image depths. Depth parameters include two ways to define color channels:

1. Each pixel has a number of "channels" of information, such as one channel for color index images, and three channels for red-green-blue (RGB) images. The **rle** format puts no restriction on the number of pixel "color channels" an image may have.

2. If a color lookup table (CLT) is present, each entry has a number of "channels" of information, such as three channels for a red-green-blue (RGB) color table (the only well-defined case). The **rle** format puts no restrictions on the number of CLT "color map channels" an image may have. Such a flexible file format design allows for the standard image types:

Color index	1 channel/pixel 3 channels/map entry	Stores color index Stores RGB value
RGB	3 channels/pixel no color map	Stores RGB value

Unfortunately, such a flexible file format can also specify a very wide range of implausible image types. So, clearly, restrictions have to be made. The image library's support of Utah **rle** image files restricts images to either 8-bit color index images (1 color channel, 3 color map color channels) or 24-bit RGB images (3 color channels, 0 or 3 color map color channels).

Reading Utah RLE files

The SDSC image library can read Utah **rle** files and map them to VFBs as follows:

Depth/color	**Mapped to VFB**
8-bit color index, with or without CLT	**IMVFBINDEX8**
8-bit color index + alpha, with or without CLT	**(IMVFBINDEX8 | IMVFBALPHA)**
24-bit RGB, with or without CLT	**IMVFBRGB**
24-bit RGB + alpha, with or without CLT	**(IMVFBRGB | IMVFBALPHA)**

If the image has a color map, the VFB includes a CLT.

Writing Utah RLE files

Images to be written out are mapped from image library VFBs as follows:

Mapped from VFB	**Depth/color**
IMVFBINDEX8	8-bit color index, with or without CLT
IMVFBRGB	24-bit RGB, with or without CLT

If the incoming VFB has a CLT, the image written to the Utah **rle** file contains a CLT.

Other image library VFB types are converted to one of the above types prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), Utah **rle** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Syntax error in incoming file
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), Utah **rle** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Design of the Utah RLE Format, Spencer W. Thomas, University of Utah, Department of Computer Science

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imrpbm - SDSC Jef Poskanzer's PBM+ suite RPBM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "rpbm", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "rpbm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "rpbm", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "rpbm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

rpbm image files are used by various tools in Jef Poskanzer's PBM+ tool suite. See the PBM documentation set for details on how to use these tools.

FILE RECOGNITION

rpbm files are recognized by the filename suffix: .rpbm.

NOTES

SDSC image library support of the **rpbm** format does not require use of the PBM+ libraries and tools. It does not contain proprietary code or any code from Jef Poskanzer's original PBM+ suite. **rpbm** format handling is available on any machine for which the SDSC image library is available.

PBM (Portable Bit Map) started with support for 1-bit monochrome images. Support was added for 8-bit grayscale images (see **impgm**(3IM)), also called PGM (Portable Grayscale Map). Support was also added for 24-bit RGB images (see **imppm**(3IM)), also called PPM (Portable Pixel Map). Each of these defined a new file format.

The original file formats were ASCII. To reduce the disk space required to store such files, three additional "raw" formats were defined. The raw formats stored the same information as their ASCII counterparts but in binary, reducing the disk space requirement by around 60%.

Today the PBM+ suite contains six related file formats:

PBM	ASCII 1-bit bitmaps
PGM	ASCII 8-bit grayscale pixel maps
PPM	ASCII 24-bit RGB color pixel maps
RPBM	Raw binary 1-bit bitmaps
RPGM	Raw binary 8-bit grayscale pixel maps
RPPM	Raw binary 24-bit RGB color pixel maps

The original PBM suite included a variety of tools. Some handled PBM files, but not PGM or PPM. Others handled PGM files, but not PBM or PPM. Then the newer PBM+ distribution introduced a set of tools that recognized all six file formats. These tools generically referred to image files as PNM (Portable aNy Map) files.

The SDSC image library treats the six PBM+ file formats separately. This **man** page only discusses the RPBM (raw monochrome) file format. The remaining file formats are discussed in their own **man** pages.

Reading RPBM image files

For compatibility with the PBM+ suite, the SDSC image library can read any of the PBM+ file formats when the **rpbm** format name is used. PBM+ files are read in and mapped to VFBs as follows:

File format	**Depth/color**	**Mapped to VFB**
PBM	1-bit monochrome	**IMVFBMONO** without a CLT
PGM	8-bit color index	**IMVFBINDEX8** without a CLT
PPM	24-bit RGB	**IMVFBRGB** without a CLT
RPBM	1-bit monochrome	**IMVFBMONO** without a CLT
RPGM	8-bit color index	**IMVFBINDEX8** without a CLT
RPPM	24-bit RGB	**IMVFBRGB** without a CLT

Header white space, and comments starting with # and extending to the end of the line are ignored. White space and comments are not allowed within the raw binary image body.

Writing RPBM image files

The SDSC image library writes **IMVFBMONO** VFBs as **rpbm** monochrome raw bitmap files.

rpbm files support no compression schemes.

Other image library VFB types are converted to **IMVFBMONO** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **rpbm** file reading returns the following error codes:

IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Premature EOF
IMESYS	System call error in read operation

In addition to those for **ImFileWrite**(3IM), **rpbm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

pbm(5), **pgm**(5), and **ppm**(5) from the PBM+ **man** page set.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **impbm**(3IM), **impgm**(3IM), **impnm**(3IM), **imppm**(3IM), **imrpgm**(3IM), **imrpnm**(3IM), **imrppm**(3IM)

AUTHORS

Dave Nadeau and Don Doering
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imrpgm - SDSC Jef Poskanzer's PBM+ suite RPGM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "rpgm", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "rpgm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "rpgm", flagsTable, dataTable )
        int         fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "rpgm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

rpgm image files are used by various tools in Jef Poskanzer's PBM+ tool suite. See the PBM documentation set for details on how to use these tools.

FILE RECOGNITION

rpgm files are recognized by the filename suffix: .rpgm.

NOTES

SDSC image library support of the **rpgm** format does not require use of the PBM+ libraries and tools. It does not contain proprietary code or any code from Jef Poskanzer's original PBM+ suite. **rpgm** format handling is available on any machine for which the SDSC image library is available.

PBM (Portable Bit Map) started with support for 1-bit monochrome images (see **impbm**(3IM)). Support was added for 8-bit grayscale images, also called PGM (Portable Grayscale Map), and 24-bit RGB images (see **imppm**(3IM)), also called PPM (Portable Pixel Map). Each of these defined a new file format.

The original file formats were ASCII. To reduce the disk space required to store such files, three additional "raw" formats were defined. The raw formats stored the same information as their ASCII counterparts but in binary, reducing the disk space requirement by around 60%.

Today the PBM+ suite contains six related file formats:

PBM	ASCII 1-bit bitmaps
PGM	ASCII 8-bit grayscale pixel maps
PPM	ASCII 24-bit RGB color pixel maps
RPBM	Raw binary 1-bit bitmaps
RPGM	Raw binary 8-bit grayscale pixel maps
RPPM	Raw binary 24-bit RGB color pixel maps

The original PBM suite included a variety of tools. Some handled PBM files, but not PGM or PPM. Others handled PGM files, but not PBM or PPM. Then the newer PBM+ distribution introduced a set of tools that recognized all six file formats. These tools generically referred to image files as PNM (Portable aNy Map) files.

The SDSC image library treats the six PBM+ file formats separately. This **man** page only discusses the RPGM (raw grayscale) file format. The remaining file formats are discussed in their own **man** pages.

Reading RPGM image files

For compatibility with the PBM+ suite, the SDSC image library can read any of the PBM+ file formats when the **rpgm** format name is used. PBM+ files are read in and mapped to VFBs as follows:

File format	Depth/color	Mapped to VFB
PBM	1-bit monochrome	**IMVFBMONO** without a CLT
PGM	8-bit color index	**IMVFBINDEX8** without a CLT
PPM	24-bit RGB	**IMVFBRGB** without a CLT
RPBM	1-bit monochrome	**IMVFBMONO** without a CLT
RPGM	8-bit color index	**IMVFBINDEX8** without a CLT
RPPM	24-bit RGB	**IMVFBRGB** without a CLT

Header white space, and comments starting with # and extending to the end of the line are ignored. White space and comments are not allowed within the raw binary image body.

Writing RPGM image files

The SDSC image library writes **IMVFBINDEX8** VFBs without CLTs as **rpgm** grayscale raw bitmap files.

rpgm files support no compression schemes.

Other image library VFB types are converted to **IMVFBINDEX8** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **rpgm** file reading returns the following error codes:

IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Premature EOF
IMESYS	System call error in read operation

In addition to those for **ImFileWrite**(3IM), **rpgm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

pbm(5), **pgm**(5), and **ppm**(5) from the PBM+ **man** page set.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **impbm**(3IM), **impgm**(3IM), **impnm**(3IM), **imppm**(3IM), **imrpbm**(3IM), **imrpnm**(3IM), **imrppm**(3IM)

AUTHORS

Dave Nadeau and Don Doering
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imrpnm - SDSC Jef Poskanzer's PBM+ suite RPNM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "rpnm", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "rpnm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "rpnm", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "rpnm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

rpnm image files are used by various tools in Jef Poskanzer's PBM+ tool suite. See the PBM documentation set for details on how to use these tools.

FILE RECOGNITION

rpnm files are recognized by the filename suffix: .rpnm.

NOTES

SDSC image library support of the **rpnm** format does not require use of the PBM+ libraries and tools. It does not contain proprietary code or any code from Jef Poskanzer's original PBM+ suite. **rpnm** format handling is available on any machine for which the SDSC image library is available.

PBM (Portable Bit Map) started with support for 1-bit monochrome images (see **impbm**(3IM)). Support was added for 8-bit grayscale images (see **impgm**(3IM)), also called PGM (Portable Grayscale Map), and 24-bit RGB images (see **imppm**(3IM)), also called PPM (Portable Pixel Map). Each of these defined a new file format.

The original file formats were ASCII. To reduce the disk space required to store such files, three additional "raw" formats were defined. The raw formats stored the same information as their ASCII counterparts but in binary, reducing the disk space requirement by around 60%.

Today the PBM+ suite contains six related file formats:

PBM	ASCII 1-bit bitmaps
PGM	ASCII 8-bit grayscale pixel maps
PPM	ASCII 24-bit RGB color pixel maps
RPBM	Raw binary 1-bit bitmaps
RPGM	Raw binary 8-bit grayscale pixel maps
RPPM	Raw binary 24-bit RGB color pixel maps

The original PBM suite included a variety of tools. Some handled PBM files, but not PGM or PPM. Others handled PGM files, but not PBM or PPM. Then the newer PBM+ distribution introduced a set of tools that recognized all six file formats. These tools generically referred to image files as PNM files, which stands for "Portable aNy Map."

The SDSC image library treats the six PBM+ file formats separately. However, to be compatible with the apparent trend in the PBM+ toolset, the new generic **pnm** and **rpnm** names mean "any of the PBM+ formats." This **man** page only discusses the RPNM generic filename. The remaining file formats are discussed in their own **man** pages.

Reading RPNM image files

The SDSC image library can read any of the PBM+ file formats when the **rpnm** format name is used, including the ASCII format variants and the raw binary variants. PBM+ files are read in and mapped to VFBs as follows:

File format	Depth/color	Mapped to VFB
PBM	1-bit monochrome	**IMVFBMONO** without a CLT
PGM	8-bit color index	**IMVFBINDEX8** without a CLT
PPM	24-bit RGB	**IMVFBRGB** without a CLT
RPBM	1-bit monochrome	**IMVFBMONO** without a CLT
RPGM	8-bit color index	**IMVFBINDEX8** without a CLT
RPPM	24-bit RGB	**IMVFBRGB** without a CLT

Header white space, and comments starting with # and extending to the end of the line are ignored. White space and comments are not allowed within the raw binary image body.

Writing RPNM image files

The SDSC image library can write PBM+ suite files in the following configurations:

Mapped from VFB	Depth/color	File format
IMVFBMONO without a CLT	1-bit monochrome	RPBM
IMVFBINDEX8 without a CLT	8-bit color index	RPGM
IMVFBRGB without a CLT	24-bit RGB	RPPM

RGB images are always stored noninterlaced (i.e., RGBRGBRGB...). Scanline- and plane-interlaced modes are not available in RPPM.

The PBM+ suite files support no compression schemes.

Other image library VFB types are converted to one of the above types prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **rpnm** file reading returns the following error codes:

IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Premature EOF
IMESYS	System call error in read operation

In addition to those for **ImFileWrite**(3IM), **rpnm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

pbm(5), **pgm**(5), and **ppm**(5) from the PBM+ **man** page set.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **impbm**(3IM), **impgm**(3IM), **impnm**(3IM), **imppm**(3IM), **imrpbm**(3IM), **imrpgm**(3IM), **imrppm**(3IM)

AUTHORS

Dave Nadeau and Don Doering
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imrppm - SDSC Jef Poskanzer’s PBM+ suite RPPM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "rppm", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "rppm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "rppm", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "rppm", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

rppm image files are used by various tools in Jef Poskanzer’s PBM+ tool suite. See the PBM documentation set for details on how to use these tools.

FILE RECOGNITION

rppm files are recognized by the filename suffix: .rppm.

NOTES

SDSC image library support of the **rppm** format does not require use of the PBM+ libraries and tools. It does not contain proprietary code or any code from Jef Poskanzer’s original PBM+ suite. **rppm** format handling is available on any machine for which the SDSC image library is available.

PBM (Portable Bit Map) started with support for 1-bit monochrome images (see **impbm**(3IM)). Support was added for 8-bit grayscale images (see **impgm**(3IM)), also called PGM (Portable Grayscale Map), and 24-bit RGB images (see **imppm**(3IM)), also called PPM (Portable Pixel Map). Each of these defined a new file format.

The original file formats were ASCII. To reduce the disk space required to store such files, three additional "raw" formats were defined. The raw formats stored the same information as their ASCII counterparts but in binary, reducing the disk space requirement by around 60%.

Today the PBM+ suite contains six related file formats:

PBM	ASCII 1-bit bitmaps
PGM	ASCII 8-bit grayscale pixel maps
PPM	ASCII 24-bit RGB color pixel maps
RPBM	Raw binary 1-bit bitmaps
RPGM	Raw binary 8-bit grayscale pixel maps
RPPM	Raw binary 24-bit RGB color pixel maps

The original PBM suite included a variety of tools. Some handled PBM files, but not PGM or PPM. Others handled PGM files, but not PBM or PPM. Then the newer PBM+ distribution introduced a set of tools that recognized all six file formats. These tools generically referred to image files as PNM (Portable aNy Map) files.

The SDSC image library treats the six PBM+ file formats separately. This **man** page only discusses the RPPM (raw RGB color pixel maps) file format. The remaining file formats are discussed in their own **man** pages.

Reading RPPM image files

For compatibility with the PBM+ suite, the SDSC image library can read any of the PBM+ file formats when the **rppm** format name is used. PBM+ files are read in and mapped to VFBs as follows:

File format	**Depth/color**	**Mapped to VFB**
PBM	1-bit monochrome	**IMVFBMONO** without a CLT
PGM	8-bit color index	**IMVFBINDEX8** without a CLT
PPM	24-bit RGB	**IMVFBRGB** without a CLT
RPBM	1-bit monochrome	**IMVFBMONO** without a CLT
RPGM	8-bit color index	**IMVFBINDEX8** without a CLT
RPPM	24-bit RGB	**IMVFBRGB** without a CLT

Header white space, and comments starting with # and extending to the end of the line are ignored. White space and comments are not allowed within the raw binary image body.

Writing RPPM image files

The SDSC image library writes **IMVFBRGB** VFBs as **rppm** RGB raw bitmap files.

rppm images are always stored noninterlaced (i.e., RGBRGBRGB...). Scanline- and plane-interlaced modes are not available in **rppm** files.

rppm files support no compression schemes.

Other image library VFB types are converted to **IMVFBRGB** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **rppm** file reading returns the following error codes:

IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Premature EOF
IMESYS	System call error in read operation

In addition to those for **ImFileWrite**(3IM), **rppm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

pbm(5), **pgm**(5), and **ppm**(5) from the PBM+ **man** page set.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM), **impbm**(3IM), **impgm**(3IM), **impnm**(3IM), **imppm**(3IM), **imrpbm**(3IM), **imrpgm**(3IM), **imrpnm**(3IM)

AUTHORS

Dave Nadeau and Don Doering
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imsynu - SDSC Synu (Synthetic Universe) file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "synu", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "synu", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "synu", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "synu", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

synu is the image file format output by SDSC's **synu** (Synthetic Universe) portable renderer.

FILE RECOGNITION

synu files are recognized only by the following filename suffix:

NOTES

SDSC image library support of the SDSC **synu** format does not require use of any other SDSC tools and contains no proprietary code. SDSC **synu** format handling is available on any machine for which the SDSC image library is available.

synu files contain "objects" of many types, such as grayscale images, RGB images, and various forms of geometry. SDSC image library translation of **synu** files is limited to the handling of grayscale and RGB objects. All other **synu** objects are ignored during reads and cannot be generated during writes.

Reading Synu files

The SDSC image library reads one or more uncompressed, 8-bit, grayscale and 24-bit, RGB **synu** images and maps them to **IMVFBINDEX8** and **IMVFBRGB** VFBs, respectively, both without color lookup tables.

Writing Synu files

Image library **IMVFBINDEX8** VFBs without color lookup tables are written as 8-bit grayscale **synu** files. IMVFBRGB VFBs without color lookup tables are written as 24-bit RGB **synu** files. Both image types are uncompressed. RGB images are noninterlaced. Scanline- and plane-interlaced modes are not supported by **synu** image files.

Other image library VFB types are converted to **IMVFBINDEX8** or **IMVFBRGB** VFBs without color lookup tables prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **synu** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), **synu** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Synu Reference Manual, San Diego Supercomputer Center

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHORS

Phil Mercurio and Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

imtiff - SDSC TIFF (Tagged Image File Format) translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "tiff", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFRead( fp, "tiff", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileWrite( fd, "tiff", flagsTable, dataTable )
        int        fd;
        TagTable   *flagsTable;
        TagTable   *dataTable;

int ImFileFWrite( fp, "tiff", flagsTable, dataTable )
        FILE       *fp;
        TagTable   *flagsTable;
        TagTable   *dataTable;
```

DESCRIPTION

tiff is a generic Tagged Image File Format developed by Aldus and Microsoft in conjunction with leading scanner and printer manufacturers. **tiff** files may contain images and miscellaneous other image-related items. Such files may be created and manipulated by a variety of Tagged Image File Format tools. See the Tagged Image File Format documentation for details on how to use these tools.

tiff support within the SDSC image library is limited to images of certain depths and storage methods.

FILE RECOGNITION

tiff files are recognized by the following filename suffixes: .tiff and .tif.

NOTES

SDSC image library support of the **tiff** format adheres to **tiff** Specification 5.0 and **tiff** Software Release 2.3. In particular **tiff** Classes B, G, P, R, are all supported for read and write with the following exceptions:

> **tiff** directories that are written do not include NewSubfileType tags. NewSubfileType indicates how one image within a **tiff** file is related to other images in the file.
>
> **tiff** directories that are written do not include ResolutionUnit tags. ResolutionUnit indicates whether the unit of pixel measure is **none, inch,** or **centimeter**.
>
> **tiff** R image files that are written do not include the new 5.0 colorimetric information tags.

Tags not recognized on reading are ignored unless the calling program has requested verbosity.

SDSC image library support of the **tiff** format does not require use of the Tagged Image File Format's **tiff** library **libtiff.a** and contains no proprietary code. **tiff** format handling is available on any machine for which the SDSC image library is available.

Reading TIFF image files

The SDSC image library can read **tiff** image files with one or more images and map them to VFBs as follows:

Depth/color	**Types of compression**	**Mapped to VFB**
1-bit monochrome	Uncompressed, LZW, Mac PackBits	IMVFBMONO
4-bit color index	Uncompressed, LZW, Mac PackBits	IMVFBINDEX8
8-bit color index	Uncompressed, LZW, Mac PackBits	IMVFBINDEX8
16-bit color index	Uncompressed, LZW, Mac PackBits	IMVFBINDEX16
24-bit RGB	Uncompressed, LZW, Mac PackBits	IMVFBRGB
32-bit RGB+Alpha	Uncompressed, LZW, Mac PackBits	(IMVFBRGB | IMVFBALPHA)

8-bit and 16-bit images may have associated color lookup tables.

RGB images may be scanline-interlaced or plane-interlaced. (In **tiff** terminology, this is called "contiguous" or "separate" planar configuration.)

Lempel-Ziv Welch compression (LZW) and Macintosh PackBit compression schemes are supported. CCITT, NEXT, THUNDERSCAN, SGIRLE, and PICIO image compression are *not* supported.

tiff files may be in MBF (most-significant byte first) or LBF (least-significant byte first) byte order. *Note:* **tiff** documentation refers to MBF as **TIFF_BIGENDIAN** and LBF as **TIFF_LITTLEENDIAN**. **TIFF_VERSION** is 42.

Bottom right and top right image orientations are not supported.

If the file's image has a color map, the image library VFB includes a color lookup table.

Note that 24-bit and 32-bit color indexes are truncated to the lower 16 bits when the file is stored as an **IMVFBINDEX16** VFB.

Writing TIFF image files

Images to be written out are mapped from image library VFBs as follows:

Mapped from VFB	**Depth/color**	**Types of compression**
IMVFBMONO	1-bit color index	Uncompressed, LZW,

		Mac PackBits
IMVFBINDEX8	8-bit color index	Uncompressed, LZW, Mac PackBits
IMVFBINDEX16	16-bit color index	Uncompressed, LZW, Mac PackBits
IMVFBRGB	24-bit RGB	Uncompressed, LZW, Mac PackBits
(IMVFBRGB|IMVFBALPHA)	32-bit RGB+Alpha	Uncompressed, LZW, Mac PackBits

If the incoming VFB has a color lookup table, the image written to the **tiff** file contains one.

RGB images may be stored in scanline-interlaced and plane-interlaced modes. The **tiff** format does not supprt noninterlaced RGB image storage.

Other image library VFB types are converted to one of the above types prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead**(3IM), **tiff** file reading returns the following error codes:

IMECONFLICT	Conflicting information in image header
IMEDEPTH	Unknown image depth
IMEHEIGHT	Zero or negative image height
IMEMAGIC	Bad magic number in image file
IMEMALLOC	Cannot allocate enough memory
IMEORIENTATION	Unsupported image orientation
IMEOUTOFRANGE	Header value out of legal range
IMEPLANES	Unknown image plane configuration
IMESYNTAX	Syntax error in **tiff** file
IMESYS	System call error in read operation
IMEUNSUPPORTED	Unsupported VFB type
IMEVERSION	Bad version number
IMEWIDTH	Zero or negative image width

In addition to those listed for **ImFileWrite**(3IM), **tiff** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

Tag Image File Format Rev 5.0, August 8, 1988, Tim Davenport, Aldus Corporation, and Manny Vellon, Microsoft Corporation.

See also the **man** pages with the **tiff** standard distribution.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

T. Todd Elvins
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

ImVfbAlloc - Allocate a virtual frame buffer

SYNOPSIS

```
#include "im.h"

ImVfb *ImVfbAlloc( width, height, fields )
        int width, height ;
        int fields ;
```

DESCRIPTION

ImVfbAlloc allocates a virtual frame buffer and returns a pointer to the new Vfb.

width and *height* are the x and y dimensions, respectively, of the desired Vfb.

fields is a bitmask that specifies what items of information need to be stored in each pixel. *fields* is formed by or'ing together one or more of:

Constant	Meaning
IMVFBRGB	Store red, green, blue values (0-255)
IMVFBALPHA	Store an alpha value (0-255)
IMVFBINDEX8	Store a color index (0-255)
IMVFBWPROT	Store a write protection (0 or non-zero)
IMVFBINDEX16	Store a color index (full integer)
IMVFBMONO	Store a monochrome (on/off) value
IMVFBZ	Store a z-value (full integer)
IMVFBIDATA	Store an integer data value
IMVFBFDATA	Store a floating-point data value

NOTES

Information about a particular virtual frame buffer can be **S**et with:

Call	Meaning
ImVfbSClt(v, c)	Set (assign) a color lookup table to the Vfb

Information about a particular virtual frame buffer can be **Q**ueried by:

Call	Meaning
ImVfbQWidth(v)	Return the number of columns
ImVfbQHeight(v)	Return the number of rows
ImVfbQFields(v)	Return the fields mask
ImVfbQNBytes(v)	Return the number of bytes per pixel
ImVfbQClt(v)	Return a pointer to a color lookup table

Frame buffer information can be allocated on a per-pixel or a per-plane basis. The per-pixel basis is better to reduce paging. The per-plane basis reduces overall storage. For a per-pixel allocation, one might call:

```
v = ImVfbAlloc( 1280, 1024, IMVFBRGB | IMVFBZ );
```

For a per-plane allocation, one might call:

```
vc = ImVfbAlloc( 1280, 1024, IMVFBRGB );
vz = ImVfbAlloc( 1280, 1024, IMVFBZ );
```

Per-pixel storage is a packed array of values. The storage convention assumes that the top row is row #0 and the left column is column #0. Pixels are stored like C-language 2D arrays: left-to-right across the row. Values can be **S**et into a particular pixel (pointed to by a pixel pointer) within a particular virtual frame buffer by:

Call	**Meaning**
ImVfbSRed(v,p,r)	Red (byte)
ImVfbSGreen(v,p,g)	Green (byte)
ImVfbSBlue(v,p,b)	Blue (byte)
ImVfbSAlpha(v,p,a)	Alpha-value (byte)
ImVfbSIndex8(v,p,i8)	Color index (byte)
ImVfbSIndex16(v,p,i32)	Color index (integer)
ImVfbSMono(v,p,m)	Monochromatic value (zero or non-zero)
ImVfbSIndex(v,p,i)	ImVfbSIndex8 or ImVfbSIndex16
ImVfbSZ(v,p,z)	Z-value (integer)
ImVfbSGray(v,p,g)	Gray scale (byte)
ImVfbSGrey(v,p,g)	Gray scale (byte)
ImVfbSFData(v,p,f)	Floating-point data value
ImVfbSIData(v,p,i)	Integer data value

Various values can be **Q**ueried within a particular virtual frame buffer by:

Call	**Meaning**
ImVfbQRed(v,p)	Red (byte)
ImVfbQGreen(v,p)	Green (byte)
ImVfbQBlue(v,p)	Blue (byte)
ImVfbQAlpha(v,p)	Alpha-value (byte)
ImVfbQIndex8(v,p)	Color index (byte)
ImVfbQIndex16(v,p)	Color index (integer)
ImVfbQMono(v,p,m)	Monochromatic value (zero or non-zero)
ImVfbQIndex(v,p)	ImVfbQIndex8 or ImVfbQIndex16
ImVfbQZ(v,p)	Z-value (integer)
ImVfbQGray(v,p)	Gray scale (byte)
ImVfbQGrey(v,p)	Gray scale (byte)
ImVfbQFData(v,p)	Floating-point data value
ImVfbQIData(v,p)	Integer data value

Pixel pointer values can be **Q**ueried and **S**et by:

Call	**Meaning**
ImVfbQPtr(v,r,c)	Return a pointer to a particular pixel
ImVfbQFirst(v)	Return a pointer to the first (UL) pixel
ImVfbQLast(v)	Return a pointer to the last (LR) pixel
ImVfbQLeft(v,p)	Return a pointer to a pixel one column left
ImVfbQRight(v,p)	Return a pointer to a pixel one column right
ImVfbQUp(v,p)	Return a pointer to a pixel one row up
ImVfbQDown(v,p)	Return a pointer to a pixel one row down
ImVfbQNext(v,p)	Same as ImVfbQRight(v,p)

ImVfbQPrev(v,p)	Same as ImVfbQLeft(v,p)
ImVfbSInc(v,p)	Same as p = ImVfbQRight(v,p)
ImVfbSDec(v,p)	Same as p = ImVfbQLeft(v,p)

The pixel just to the **ImVfbQRight** of the right-most pixel in a scanline is the left-most pixel in the next scanline. The pixel just to the **ImVfbQLeft** of the left-most pixel in a scanline is the last pixel in the previous scanline. No automatic wraparound occurs between the last pixel and the first pixel in the framebuffer.

RETURNS

An error causes **IMVFBNULL** to be returned and the value of **ImErrNo** to be set to one of the following:

IMEMALLOC	Cannot allocate enough memory for the new Vfb
IMENOFIELDS	The *fields* mask is zero

SEE ALSO

imintro (3IM), **ImVfbFree** (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImVfbCopy - Copy a subarea within a virtual frame buffer

SYNOPSIS

```
#include "im.h"

ImVfb *ImVfbCopy( srcVfb, srcXLeft, srcYTop, srcDX, srcDY, fieldMask, dstVfb, dstXLeft, dstYTop )
        ImVfb *srcVfb ;
        int srcXLeft, srcYTop ;
        int srcDX, srcDY ;
        int fieldMask ;
        ImVfb *dstVfb ;
        int dstXLeft, dstYTop ;
```

DESCRIPTION

ImVfbCopy copies a portion of a virtual frame buffer to another virtual frame buffer. A pointer to the destination virtual frame buffer is returned.

srcVfb is the Vfb whose subarea is to be copied.

srcXLeft, srcYTop indicate the top-left corner of the area of the Vfb to be copied.

srcDX, srcDY are the dimensions of the subarea of the Vfb to be copied.

fieldMask is a mask of field constants (see **ImVfbAlloc** (3IM)) that selects the fields in *srcVfb* to be copied into *dstVfb*.

dstVfb is the Vfb to receive the copied area.

dstXLeft, dstYTop indicate the top-left corner of the area to which the subarea will be copied.

NOTES

If *dstVfb* is the constant **IMVFBNEW**, a new Vfb is allocated for the copied data. The Vfb has the same size as the region being copied, and the fields are selected by *fieldMask*. A pointer to the new Vfb is returned.

Portions of the copied data that would extend beyond the borders of the *dstVfb* are skipped.

The *srcVfb* and *dstVfb* may be the same Vfb. The source and destination areas can overlap with no unpleasant side effects.

RETURNS

Upon success, **ImVfbCopy** returns a pointer to the destination Vfb. Upon failure, **IMVFBNULL** is returned and **ImErrNo** set to the following:

IMEMALLOC	Cannot allocate enough memory for the new Vfb

SEE ALSO

ImVfbAlloc (3IM), **ImVfbDup** (3IM)

AUTHORS

Mike Bailey, Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

ImVfbDup - Duplicate a virtual frame buffer

SYNOPSIS

```
#include "im.h"

ImVfb *ImVfbDup( srcVfb )
        ImVfb *srcVfb ;
```

DESCRIPTION

ImVfbDup duplicates a virtual frame buffer and returns a pointer to the new Vfb. *srcVfb* is the Vfb to be duplicated. It is not changed by this operation.

RETURNS

An error causes **IMVFBNULL** to be returned and the value of **ImErrNo** to be set to the following:

IMEMALLOC	Cannot allocate enough memory for the new Vfb

SEE ALSO

ImVfbAlloc (3IM), **ImVfbCopy** (3IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImVfbFlip - Flip a virtual frame buffer vertically, horizontally, or both

SYNOPSIS

```
#include "im.h"

ImVfb *ImVfbFlip( srcVfb, flipDirection, dstVfb )
        ImVfb *srcVfb ;
        int flipDirection ;
        ImVfb *dstVfb ;
```

DESCRIPTION

ImVfbFlip flips a virtual frame buffer about the horizontal and/or vertical axis and returns a pointer to the flipped Vfb.

srcVfb is the Vfb to be flipped.

flipDirection is the direction(s) about which to flip. Possible values for *flipDirection* are:

Value	Meaning
IMVFBXFLIP	Flip in X
IMVFBYFLIP	Flip in Y
IMVFBXYFLIP	Flip in X and Y

Note that specifying **IMVFBXYFLIP** causes a complete pixel transpose.

dstVfb is the Vfb to contain the flipped image. If *dstVfb* is the constant **IMVFBNEW**, a new Vfb is allocated with the same size and the same fields as *srcVfb*. A pointer to the new Vfb is returned.

RETURNS

Upon success, **ImVfbDup** returns a pointer to the destination Vfb. Upon failure, **IMVFBNULL** is returned and **ImErrNo** set to one of the following:

IMEMALLOC	Cannot allocate enough memory for the new Vfb
IMEBADFLIP	*flipDirection* is not one of the legal values

SEE ALSO

ImVfbResize (3IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImVfbFree - Free the storage for a virtual frame buffer

SYNOPSIS

```
#include "im.h"

void ImVfbFree( srcVfb )
        ImVfb *srcVfb ;
```

DESCRIPTION

ImVfbFree frees the dynamic storage for a virtual frame buffer.

srcVfb is the Vfb whose storage is to be freed. *srcVfb* is no longer valid after this call.

NOTES

ImVfbFree does not free the storage of any **ImClt** that might be attached to it.

SEE ALSO

ImCltAlloc (3IM), **ImCltFree** (3IM), **ImVfbAlloc** (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImVfbQClt - Query the CLT (color lookup table) that is attached to a virtual frame buffer
ImVfbSClt - Set the CLT attached to a virtual frame buffer

SYNOPSIS

```
#include "im.h"

ImClt *ImVfbQClt( srcVfb )
        ImVfb *srcVfb ;

void ImVfbSClt( srcVfb, clt )
        ImVfb *srcVfb ;
        ImClt *clt ;
```

DESCRIPTION

ImVfbSClt attaches the color lookup table *clt* to the virtual frame buffer *srcVfb*. Thereafter, color index information in the virtual frame buffer is looked up in *clt* to obtain RGB color triplets.

ImVfbQClt returns a pointer to the color lookup table attached to the *srcVfb*. A return value of **IMCLTNULL** indicates the Vfb currently has no color lookup table.

NOTES

Setting a Vfb's CLT to **IMCLTNULL** unattaches any CLT from the Vfb.

Both **ImVfbSClt** and **ImVfbQClt** are macros.

RETURNS

ImVfbQClt returns the *srcVfb*'s CLT.

ImVfbSClt returns nothing.

SEE ALSO

ImCltAlloc (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

LIBRARY"

NAME

ImVfbQFields - Query the pixel fields of a virtual frame buffer

SYNOPSIS

#include "im.h"

int ImVfbQFields(srcVfb)
    ImVfb *srcVfb ;

DESCRIPTION

ImVfbQFields returns a bitmask specifying the fields stored in *srcVfb*.

NOTES

fields specifies what information is to be stored in each pixel of the Vfb. *fields* is a bitmask formed by or'ing together one or more of the same constants used by the *fields* parameter in **ImVfbAlloc** (3IM).

ImVfbQFields is a macro.

RETURNS

ImVfbQFields returns the fields mask of the *srcVfb*.

SEE ALSO

ImVfbAlloc (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

LIBRARY"

NAME

ImVfbQNBytes - Query the number of bytes per pixel being stored in a virtual frame buffer
ImVfbQWidth - Query the width of a virtual frame buffer
ImVfbQHeight - Query the height of a virtual frame buffer

SYNOPSIS

```
#include "im.h"

int ImVfbQNBytes( srcVfb )
        ImVfb *srcVfb ;

int ImVfbQWidth( srcVfb )
        ImVfb *srcVfb ;

int ImVfbQHeight( srcVfb )
        ImVfb *srcVfb ;
```

DESCRIPTION

ImVfbQNBytes returns the number of bytes per pixel used by the *srcVfb*.

ImVfbQWidth returns the width of the *srcVfb* in pixels.

ImVfbQHeight returns the height of the *srcVfb* in pixels.

NOTES

All are C macros.

SEE ALSO

ImVfbAlloc (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImVfbQPtr - Query the pointer to a pixel location in a virtual frame buffer
ImVfbQFirst - Query the pointer to the first pixel location in a virtual frame buffer
ImVfbQLast - Query the pointer to the last pixel location in a virtual frame buffer
ImVfbQLeft - Query the pointer to the next pixel to the left in a virtual frame buffer
ImVfbQRight - Query the pointer to the next pixel to the right in a virtual frame buffer
ImVfbQUp - Query the pointer to the next pixel up in a virtual frame buffer
ImVfbQDown - Query the pointer to the next pixel down in a virtual frame buffer
ImVfbQNext - Query the pointer to the next pixel in a virtual frame buffer
ImVfbQPrev - Query the pointer to the previous pixel in a virtual frame buffer
ImVfbSInc - Increment the pointer to point to the next pixel in a virtual frame buffer
ImVfbSDec - Decrement the pointer to point to the previous pixel in a virtual frame buffer

SYNOPSIS

```
#include "im.h"

ImVfbPtr ImVfbQPtr( srcVfb, x, y )
        ImVfb *srcVfb ;
        int x, y ;

ImVfbPtr ImVfbQFirst( srcVfb )
        ImVfb *srcVfb ;

ImVfbPtr ImVfbQLast( srcVfb )
        ImVfb *srcVfb ;

ImVfbPtr ImVfbQLeft( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

ImVfbPtr ImVfbQRight( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

ImVfbPtr ImVfbQUp( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

ImVfbPtr ImVfbQDown( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

ImVfbPtr ImVfbQNext( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

ImVfbPtr ImVfbQPrev( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
```

```
void ImVfbSInc( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

void ImVfbSDec( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
```

DESCRIPTION

ImVfbQPtr returns the pointer to the *srcVfb* pixel at coordinate (*x*,*y*). (0,0) is the upper left corner of the image.

ImVfbQFirst returns the pointer to the first pixel in the *srcVfb* and is equivalent to **ImVfbQPtr(srcVfb, 0, 0)**.

ImVfbQLast returns the pointer to the last pixel in the *srcVfb* and is equivalent to **ImVfbQPtr(srcVfb, ImVfbQWidth(srcVfb), ImVfbQHeight(srcVfb))**.

ImVfbQLeft returns the pointer to the pixel in the *srcVfb*, just to the left of the pixel pointed to by *p*. If *p* points to the left-most pixel of a scanline, **ImVfbQLeft** wraps around to the end of the previous scanline.

ImVfbQRight returns the pointer to the pixel in the *srcVfb*, just to the right of the pixel pointed to by *p*. If *p* points to the right-most pixel of a scanline, **ImVfbQRight** wraps around to the start of the next scanline.

ImVfbQUp returns the pointer to the pixel in the *srcVfb*, which is just above the pixel pointed to by *p*.

ImVfbQDown returns the pointer to the pixel in the *srcVfb*, which is just below the pixel pointed to by *p*.

ImVfbQNext returns the pointer to the pixel in the *srcVfb* just to the right of the pixel pointed to by *p* (identical to **ImVfbQRight**).

ImVfbQPrev returns the pointer to the pixel in the *srcVfb* just to the left of the pixel pointed to by *p* (identical to **ImVfbQLeft**).

ImVfbSInc increments pointer *p* by one and is equivalent to **p = ImVfbQNext(srcVfb, p)**.

ImVfbSDec decrements pointer *p* by one and is equivalent to **p = ImVfbQPrev(srcVfb, p)**.

NOTES

The pixel just to the **ImVfbQRight** of the right-most pixel in a scanline is the left-most pixel in the next scanline down. The pixel just to the **ImVfbQLeft** of the left-most pixel in a scanline is the last pixel in the previous scanline.

No automatic wraparound occurs between the last pixel and the first pixel in the frame buffer.

All are C macros.

SEE ALSO

ImVfbAlloc (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

ImVfbQRed - Query the red value of a pixel in a virtual frame buffer
ImVfbQGreen - Query the green value of a pixel in a virtual frame buffer
ImVfbQBlue - Query the blue value of a pixel in a virtual frame buffer
ImVfbQAlpha - Query the alpha value of a pixel in a virtual frame buffer
ImVfbQIndex8 - Query the 8-bit index value of a pixel in a virtual frame buffer
ImVfbQIndex16 - Query the 32-bit index value of a pixel in a virtual frame buffer
ImVfbQIndex - Query the index value of a pixel in a virtual frame buffer
ImVfbQZ - Query the z-buffer value of a pixel in a virtual frame buffer
ImVfbQGray, ImVfbQGrey - Query the gray scale value of a pixel in a virtual frame buffer
ImVfbQFData - Query the floating-point data value of a pixel in a virtual frame buffer
ImVfbQIData - Query the integer data value of a pixel in a virtual frame buffer
ImVfbSRed - Set the red value of a pixel in a virtual frame buffer
ImVfbSGreen - Set the green value of a pixel in a virtual frame buffer
ImVfbSBlue - Set the blue value of a pixel in a virtual frame buffer
ImVfbSAlpha - Set the alpha value of a pixel in a virtual frame buffer
ImVfbSIndex8 - Set the 8-bit index value of a pixel in a virtual frame buffer
ImVfbSIndex16 - Set the 32-bit index value of a pixel in a virtual frame buffer
ImVfbSIndex - Set the index value of a pixel in a virtual frame buffer
ImVfbSZ - Set the z-buffer value of a pixel in a virtual frame buffer
ImVfbSGray, ImVfbSGrey - Set the gray scale value of a pixel in a virtual frame buffer
ImVfbSFData - Set the floating-point data value of a pixel in a virtual frame buffer
ImVfbSIData - set the integer data value of a pixel in a virtual frame buffer

SYNOPSIS

```
#include "im.h"

int ImVfbQRed( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

int ImVfbQGreen( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

int ImVfbQBlue( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

int ImVfbQAlpha( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

int ImVfbQIndex8( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

int ImVfbQIndex16( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
```

```
int ImVfbQIndex( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

int ImVfbQZ( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

int ImVfbQGray( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

int ImVfbQGrey( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

float ImVfbQFData( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        float f ;

int ImVfbQIData( srcVfb, p )
        ImVfb *srcVfb ;
        ImVfbPtr p ;

void ImVfbSRed( srcVfb, p, r )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int r ;

void ImVfbSGreen( srcVfb, p, g )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int g ;

void ImVfbSBlue( srcVfb, p, b )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int b ;

void ImVfbSAlpha( srcVfb, p, a )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int a ;

void ImVfbSIndex8( srcVfb, p, i8 )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int i8 ;
```

```
void ImVfbSIndex16( srcVfb, p, i32 )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int i32 ;

void ImVfbSIndex( srcVfb, p, i )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int i ;

void ImVfbSZ( srcVfb, p, z )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int z ;

void ImVfbSGray( srcVfb, p, g )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int g ;

void ImVfbSGrey( srcVfb, p, g )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int g ;

void ImVfbSFData( srcVfb, p, f )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        float f ;

void ImVfbSIData( srcVfb, p, i )
        ImVfb *srcVfb ;
        ImVfbPtr p ;
        int i ;
```

DESCRIPTION

These routines store information into and query information from a pixel (pointed to by *p*) in Vfb *srcVfb*.

ImVfbSRed, **ImVfbSGreen**, and **ImVfbSBlue** set RGB information into a Vfb. They are only valid if the Vfb was **ImVfbAlloc**ed with **IMVFBRGB** in the field mask. **ImVfbQRed**, **ImVfbQGreen**, and **ImVfbQBlue** query the RGB information.

ImVfbSAlpha sets an alpha-channel value. It is only valid if the Vfb was **ImVfbAlloc**ed with **IMVFBALPHA** in the field mask. **ImVfbQAlpha** queries the alpha-channel value.

ImVfbSIndex8 sets an 8-bit **ImClt** index. It is only valid if the Vfb was **ImVfbAlloc**ed with **IMVFBINDEX8** in the field mask. **ImVfbQIndex8** queries the 8-bit **ImClt** index.

ImVfbSIndex16 sets a 16-bit **ImClt** index. It is only valid if the Vfb was **ImVfbAlloc**ed with **IMVFBINDEX16** in the field mask. **ImVfbQIndex16** queries the 16-bit **ImClt** index.

ImVfbSIndex sets an 8- or 16-bit **ImClt** index. It is only valid if the Vfb was **ImVfbAlloc**ed with **IMVFBINDEX8** or **IMVFBINDEX16** in the field mask. **ImVfbSIndex** uses whichever type of index is being used in *srcVfb*. **ImVfbQIndex** queries the **ImClt** index.

ImVfbSZ sets the z-buffer value. It is only valid if the Vfb was **ImVfbAlloc**ed with **IMVFBZ** in the field mask. **ImVfbQZ** queries the z-buffer value.

ImVfbSGray and **ImVfbSGrey** are identical to **ImVfbSIndex8**, and **ImVfbQGray** and **ImVfbQGrey** are identical to **ImVfbQIndex8**.

ImVfbSFData sets an arbitrary floating-point value. It is only valid if the Vfb was **ImVfbAlloc**ed with **IMVFBFDATA** in the field mask. **ImVfbQFData** queries the floating-point value.

ImVfbSIData sets an arbitrary integer value. It is only valid if the Vfb was **ImVfbAlloc**ed with **IMVFBIDATA** in the field mask. **ImVfbQIData** queries the integer value.

NOTES

All of these routines are C macros.

SEE ALSO

ImVfbAlloc (3IM)

AUTHOR

Mike Bailey
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

ImVfbResize - Change the resolution of a virtual frame buffer

SYNOPSIS

```
#include "im.h"

ImVfb *ImVfbResize( srcVfb, algorithm, dstVfb, width, height )
        ImVfb *srcVfb ;
        int algorithm ;
        ImVfb *dstVfb ;
        int width, height ;
```

DESCRIPTION

ImVfbResize changes the resolution of a virtual frame buffer and returns a pointer to the resized destination Vfb.

srcVfb is the Vfb to be copied and resized. *srcVfb* is unaltered by the operation.

width and *height* are the dimensions of the destination resized Vfb.

algorithm is what algorithm to use to change the resolution. Legal values of *algorithm* are:

Value	Meaning
IMVFBBILINEAR	Perform bilinear interpolation
IMVFBPIXELREP	Perform pixel replication

dstVfb is the Vfb to contain the resized image. If *dstVfb* is the constant **IMVFBNEW**, a new Vfb is allocated as *width x height* pixels with the same fields as the *srcVfb*. The new Vfb is returned.

If *dstVfb* is not **IMVFBNEW**, it must be *width x height* in size and have the same fields as *srcVfb*.

NOTES

The new Vfb contains the same picture as did the original, but it is represented with more or fewer pixels than the original.

RETURNS

Upon success, **ImVfbResize** returns a pointer to the destination Vfb. Upon failure, **IMVFBNULL** is returned, and **ImErrNo** set to one of the following:

IMEMALLOC	Cannot allocate enough memory for the new Vfb
IMEBADALGORITHM	A legal algorithm was not specified

SEE ALSO

ImVfbFlip (3IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

ImVfbToIndex8 - Convert a virtual frame buffer to an 8-bit index image
ImVfbToIndex16 - Convert a virtual frame buffer to a 16-bit index image
ImVfbToRgb - Convert a virtual frame buffer to an RGB image
ImVfbToGray - Convert a virtual frame buffer to a grayscale image
ImVfbToGrey - Convert a virtual frame buffer to a grayscale image
ImVfbToMono - Convert a virtual frame buffer to a monochrome image

SYNOPSIS

```
#include "im.h"

ImVfb *ImVfbToIndex8( srcVfb, dstVfb )
        ImVfb *srcVfb ;
        ImVfb *dstVfb ;

ImVfb *ImVfbToIndex16( srcVfb, dstVfb )
        ImVfb *srcVfb ;
        ImVfb *dstVfb ;

ImVfb *ImVfbToRgb( srcVfb, dstVfb )
        ImVfb *srcVfb ;
        ImVfb *dstVfb ;

ImVfb *ImVfbToGray( srcVfb, dstVfb )
        ImVfb *srcVfb ;
        ImVfb *dstVfb ;

ImVfb *ImVfbToGrey( srcVfb, dstVfb )
        ImVfb *srcVfb ;
        ImVfb *dstVfb ;

ImVfb *ImVfbToMono( srcVfb, threshold, dstVfb )
        ImVfb *srcVfb ;
        int threshold ;
        ImVfb *dstVfb ;
```

DESCRIPTION

Each of these routines convert an image stored in *srcVfb* into an image of a different type and store it in *dstVfb*. A pointer to the converted destination Vfb is returned.

ImVfbToIndex8 converts a virtual frame buffer to an 8-bit image.

ImVfbToIndex16 converts a virtual frame buffer to a 16-bit image.

ImVfbToRgb converts a virtual frame buffer to an RGB image.

ImVfbToGray and **ImVfbToGrey** convert a virtual frame buffer to an 8-bit gray scale image.

ImVfbToMono converts a virtual frame buffer to a monochrome image. Each monochromatic pixel value, queried by **ImVfbQMono** (3IM), is 0 (white) or 1 (black).

NOTES

The *srcVfb* and *dstVfb* may be the same Vfb without unpleasant side effects.

When **ImVfbToGray** is called, the resulting gray scale values are placed in the **IMVFBINDEX8** field of the destination virtual frame buffer.

Conversion from grayscale to monochrome uses the *threshold* value as the breakpoint between white and black values. Grayscale pixels with values less than the threshold are converted to white; those equal to or greater than the threshold are converted to black.

Conversion from color to grayscale computes the gray value for each pixel using the NTSC Y equation:

$$Gray = 0.30*R + 0.59*G + 0.11*B$$

Conversion from a color index image to an RGB image uses the *srcVfb*'s color lookup table to look up each pixel value to obtain its RGB color. If the *srcVfb* has no color lookup table, a grayscale ramp is used.

Conversion from an RGB image to a color index image scans the RGB image to build a color lookup table. **ImVfbToIndex8** builds a new 256-entry (or fewer) color lookup table, while **ImVfbToIndex16** builds a 65536-entry (or fewer) color lookup table. If the RGB image uses more colors than may be stored in such color tables, then color approximations are made to minimize color oddities.

RETURNS

Upon success, all functions return a pointer to the converted destination Vfb. An error causes **IMVFBNULL** to be returned and the value of **ImErrNo** to be set to one of the following:

IMEMALLOC	Cannot allocate enough memory for the new Vfb
IMENOTINFO	There is not enough information in *srcVfb* for the operation

SEE ALSO

ImVfbAlloc (3IM)

AUTHORS

Mike Bailey, Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

imx - SDSC AVS X file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "x", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFRead( fp, "x", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileWrite( fd, "x", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFWrite( fp, "x", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;
```

DESCRIPTION

x image files are generated by Stardent Computer, Inc.'s Application Visualization System (AVS). See the AVS documentation set for details on how to use these tools.

FILE RECOGNITION

AVS **x** files are recognized by the following filename suffixes: .x and .avs.

NOTES

SDSC image library support of the AVS **x** format does not require use of any AVS tools or Stardent hardware and contains no proprietary code. AVS **x** format handling is available on any machine for which the SDSC image library is available.

Reading AVS X files

The SDSC image library reads uncompressed, noninterlaced, 32-bit RGB-Alpha AVS **x** files and maps them to **(IMVFBRGB | IMVFBALPHA)** VFBs without color lookup tables (CLTs).

Writing AVS X files

Image library **IMVFBRGB** and **(IMVFBRGB | IMVFBALPHA)** VFBs are written to uncompressed, noninterlaced, 32-bit RGB-Alpha AVS **x** files. The AVS **x** format does not support scanline- and plane-interlaced modes.

Other image library VFB types are converted to **IMVFBRGB** VFBs prior to being written out. See the **ImFileWrite** (3IM) **man** page for details.

ERRORS

In addition to those listed for **ImFileRead** (3IM), AVS **x** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite** (3IM), AVS **x** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

AVS User's Guide, Appendix E, p. E-3, Stardent Computer Inc.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

NAME

imxbm - SDSC X11 XBM file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "xbm", flagsTable, dataTable )
        int       fd;
        TagTable  *flagsTable;
        TagTable  *dataTable;

int ImFileFRead( fp, "xbm", flagsTable, dataTable )
        FILE      *fp;
        TagTable  *flagsTable;
        TagTable  *dataTable;

int ImFileWrite( fd, "xbm", flagsTable, dataTable )
        int       fd;
        TagTable  *flagsTable;
        TagTable  *dataTable;

int ImFileFWrite( fp, "xbm", flagsTable, dataTable )
        FILE      *fp;
        TagTable  *flagsTable;
        TagTable  *dataTable;
```

DESCRIPTION

xbm bitmap image files are generated by MIT's X Window System, version 11 (hereafter referred to as X11). **xbm** files are generated by the X11 **bitmap**(1) bitmap editor and used by most X11 tools to define cursors, icons, and other monochrome glyphs. See the X11 documentation set for details on how to use tools and subroutines dealing with X11 bitmaps.

FILE RECOGNITION

X11 **xbm** files are recognized by the following filename suffixes: .xbm and .bm.

NOTES

SDSC image library support of the X11 **xbm** format does not require use of any X11 libraries and contains no proprietary code. X11 **xbm** format handling is available on any machine for which the SDSC image library is available.

xbm files contain simple C code with two **#define**'s for the width and height of the image, two optional **#define**'s for the X and Y location of the cursor hot spot, and one static character array declaration and initialization for the image bits as follows:

```
#define name_width  xxx
#define name_height yyy
#define name_x_hot  xhot
#define name_y_hot  yhot
static char name_bits[ ] = {
        ... ASCII hex data ...
        };
```

xxx and *yyy* define the width and height of the image stored in the static character array.

xhot and *yhot* define the hot spot location if the image is to be used as a cursor.

The array is initialized with hex byte values, with each byte holding 8 1-bit pixel values.

Reading X11 XBM files

The SDSC image library reads X11 **xbm** bitmap glyphs and maps them to **IMVFBMONO** VFBs without color lookup tables (CLTs).

xbm hot spot locations, if present, are ignored.

Writing X11 XBM files

The SDSC image library writes **IMVFBMONO** VFBs as X11 **xbm** bitmap glyphs.

Other image library VFB types are converted to **IMVFBMONO** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for details.

Image width and height **#define**'s are based on the VFB's width and height.

Warning: The X11 bitmap editor **bitmap**(1) was designed to edit small glyphs, like icons and cursors. It tends to be have severe difficulty with large bitmaps, such as those that can be generated using the SDSC image library.

Hot spot **#define**'s are not output.

The *name* portion of the **#define** and static array names are determined as follows:

If the filename tag is not present in the **flagsTable** argument to **ImFileWrite**(3IM), *name* defaults to **stream**.

If the filename tag is present, the name is stripped of any leading path (up to the last /). Characters up to the first character not in the set 0-9a-zA-Z_ are used to construct the name. If the first character is numeric, a leading x is prepended. For example,

flagsTable filename	**Name replacement**
myfile.xbm	myfile
/this/that/the_other.xbm	the_other
42.xbm	x42
blurtˆ&#$$.xbm	blurt

ERRORS

In addition to those listed for **ImFileRead**(3IM), X11 **xbm** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Syntax error in parsing **xbm** file
IMESYS	System call error in read operation

In addition to those listed for **ImFileWrite**(3IM), X11 **xbm** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

bitmap(1) from the X11 **man** page set.

Xlib - C Language X Interface, MIT X Window System, Version 11.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**

Notes

NAME

imxwd - SDSC X11 XWD file translation

SYNOPSIS

```
#include <stdio.h>
#include "sdsc.h"
#include "im.h"

int ImFileRead( fd, "xwd", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFRead( fp, "xwd", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileWrite( fd, "xwd", flagsTable, dataTable )
        int         fd;
        TagTable    *flagsTable;
        TagTable    *dataTable;

int ImFileFWrite( fp, "xwd", flagsTable, dataTable )
        FILE        *fp;
        TagTable    *flagsTable;
        TagTable    *dataTable;
```

DESCRIPTION

xwd window dump image files are used by the **xwd**(1) and **xwud**(1) tools of MIT's X Window System, version 11 (hereafter referred to as X11). See the X11 documentation set for details on how to use these tools.

FILE RECOGNITION

X11 **xwd** files are recognized by the following filename suffixes: .xwd and .x11.

NOTES

SDSC image library support of the X11 **xwd** format does not require use of any X11 libraries and contains no proprietary code. X11 **xwd** format handling is available on any machine for which the SDSC image library is available.

Reading X11 XWD files

The SDSC image library supports reading of **xwd** files that have the following file format features:

Version 7 format (X Window system, version 11).

Z-format pixmaps only.

8-bit color index images with or without color maps and 24-bit RGB images with or without color maps.

Pixel values stored in 8-bit, 16-bit, or 32-bit bitmap units in MBF (most-significant byte first) or LBF (least-significant byte first) byte order.

The SDSC image library does not support reading of **xwd** files that have the following file format features:

Version 6 format or older (obsolete X Window System, version 10).

XY bitmaps or XY pixmaps. Primarily used for monochrome images.

xwd fields meant for use to display the image on an X display are ignored. These fields include the window width, height, and (x,y) location; and border width.

Incoming 8-bit **xwd** images are mapped to image library **IMVFBINDEX8** VFBs. If the **xwd** file includes a color map, the VFB contains a color lookup table (CLT).

Incomming 24-bit **xwd** images are mapped to image library **IMVFBRGB** VFBs. If the **xwd** file includes a color map (DirectColor visual class), the color map is automatically applied to incoming RGB data as pixels are stored into the VFB. The returned VFB will not contain the **xwd** file's color map.

Writing X11 XWD files

The SDSC image library writes **IMVFBINDEX8** and **IMVFBRGB** VFBs as X11 **xwd** image files. **IMVFBINDEX8** VFBs with a CLT are written including a color map.

Other image library VFB types are converted to **IMVFBINDEX8** or **IMVFBRGB** VFBs prior to being written out. See the **ImFileWrite**(3IM) **man** page for further details.

xwd files written by the SDSC image library are always version 7 (X11), Z-format pixmaps. 8-bit images are written with 8-bits per pixel and an 8-bit bitmap unit. 24-bit images are written with 32-bits per pixel and a 32-bit bitmap unit. Byte and bit order are always MBF (most-significant byte first).

Window display attributes are set to use a pseudo-color (8-bit) or true-color (24-bit) visual class, a window height and width the same as the image height and width, a window (x,y) location of (0,0) and a 0 border width. The window name is set to **xwdump** if no "image name" tag is found in the **dataTable**.

ERRORS

In addition to those listed for **ImFileWrite**(3IM), X11 **xwd** file reading returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYNTAX	Syntax error in parsing **xwd** file
IMESYS	System call error in read operation
IMEUNSUPPORTED	Unsupported feature of **xwd** file format

In addition to those listed for **ImFileWrite**(3IM), X11 **xwd** file writing returns the following error codes:

IMEMALLOC	Cannot allocate enough memory
IMESYS	System call error in write operation

DOCUMENTATION

xwd(1) and **xwud**(1) from the X11 **man** page set.

Xlib - C Language X Interface, MIT X Window System, Version 11.

SEE ALSO

imconv(1IM), **imfile**(1IM), **imformats**(1IM)

AUTHOR

Dave Nadeau
San Diego Supercomputer Center

CONTACT

SDSC consultants, (619)534-5100, **consult@y1.sdsc.edu**