PAWT File Format Documentation

FLIR FPF Format

https://exiftool.org/TagNames/FLIR.html

Index1	Tag Name	Writable	Values / Notes
32	FPFVersion	no	
36	ImageDataOffset	no	
40	ImageType	no	0 = Temperature 1 = Temperature Difference 2 = Object Signal 3 = Object Signal Difference
42	ImagePixelFormat	no	0 = 2-byte short integer 1 = 4-byte long integer 2 = 4-byte float 3 = 8-byte double
44	ImageWidth	no	
46	ImageHeight	no	
48	ExternalTriggerCount	no	
52	SequenceFrameNumber	no	
120	CameraModel	no	
152	CameraPartNumber	no	
184	CameraSerialNumber	no	
216	CameraTemperatureRangeMin	no	
220	CameraTemperatureRangeMax	no	
224	LensModel	no	
256	LensPartNumber	no	
288	LensSerialNumber	no	
320	FilterModel	no	
336	FilterPartNumber	no	
384	FilterSerialNumber	no	
480	Emissivity	no	
484	ObjectDistance	no	
488	ReflectedApparentTemperature	no	
492	AtmosphericTemperature	no	
496	RelativeHumidity	no	
500	ComputedAtmosphericTrans	no	
504	EstimatedAtmosphericTrans	no	
508	ReferenceTemperature	no	
512	IRWindowTemperature	no	
516	IRWindowTransmission	no	
584	DateTimeOriginal	no	
676	CameraScaleMin	no	
680	CameraScaleMax	no	
684	CalculatedScaleMin	no	
688	CalculatedScaleMax	no	
692	ActualScaleMin	no	
696	ActualScaleMax	no	

FITS File Format

https://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html

https://heasarc.gsfc.nasa.gov/docs/software/ftools/fv/fitsTcl.html

PGM / PPM Format

http://netpbm.sourceforge.net/doc/pbm.html

http://netpbm.sourceforge.net/doc/ppm.html

NAME

pbm - Netpbm bi-level image format

DESCRIPTION

This program is part of Netpbm.

The PBM format is a lowest common denominator monochrome file format. It serves as the common language of a large family of bitmap image conversion filters. Because the format pays no heed to efficiency, it is simple and general enough that one can easily develop programs to convert to and from just about any other graphics format, or to manipulate the image.

The name "PBM" is an acronym derived from "Portable Bit Map."

This is not a format that one would normally use to store a file or to transmit it to someone -- it's too expensive and not expressive enough for that. It's just an intermediary format. In it's purest use, it lives only in a pipe between two other programs.

THE LAYOUT

The format definition is as follows.

A PBM file consists of a sequence of one or more PBM images. There are no data, delimiters, or padding before, after, or between images.

Each PBM image consists of the following:

- A "magic number" for identifying the file type. A pbm image's magic number is the two characters "P4".
- Whitespace (blanks, TABs, CRs, LFs).
- The width in pixels of the image, formatted as ASCII characters in decimal.
- Whitespace.
- The height in pixels of the image, again in ASCII decimal.
- A single whitespace character (usually a newline).
- A raster of Height rows, in order from top to bottom. Each row is Width bits, packed 8 to a byte, with don't care bits to fill out the last byte in the row. Each bit represents a pixel: 1 is black, 0 is white. The order of the pixels is left to right. The order of their storage within each file byte is most significant bit to least significant bit. The order of the file bytes is from the beginning of the file toward the end of the file.

  A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

- Before the whitespace character that delimits the raster, any characters from a "#" through the next carriage return or newline character, is a comment and is ignored. Note that this is rather unconventional, because a comment can actually be in the middle of what you might consider a token. Note also that this means if you have a comment right before the raster, the newline at the end of the comment is not sufficient to delimit the raster.

All characters referred to herein are encoded in ASCII. "newline" refers to the character known in ASCII as Line Feed or LF. A "white space" character is space, CR, LF, TAB, VT, or FF (I.e. what the ANSI standard C isspace() function calls white space).

Plain PBM

There is actually another version of the PBM format, even more simplistic, more lavishly wasteful of space than PBM, called Plain PBM. Plain PBM actually came first, but even its inventor couldn't stand its recklessly squanderous use of resources after a while and switched to what we now know as the regular PBM format. But Plain PBM is so redundant -- so overstated -- that it's virtually impossible to break. You can send it through the most liberal mail system (which was the original purpose of the PBM format) and it will arrive still readable. You can flip a dozen random bits and easily piece back together the original image. And we hardly need to define the format here, because you can decode it by inspection.

Netpbm programs generate Raw PBM format instead of Plain PBM by default, but the common option **-plain** chooses Plain PBM.

The difference is:

- There is exactly one image in a file.
- The "magic number" is "P1" instead of "P4".
- Each pixel in the raster is represented by a byte containing ASCII '1' or '0', representing black and white respectively. There are no fill bits at the end of a row.
- White space in the raster section is ignored.
- You can put any junk you want after the raster, if it starts with a white space character.
- No line should be longer than 70 characters.

Here is an example of a small image in the plain PBM format.

```
P1
# feep.pbm
24 7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0
0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

There is a newline character at the end of each of these lines.

You can generate the Plain PBM format from the regular PBM format (first image in the file only) with the **pnmtoplainpnm** program.

Programs that read this format should be as lenient as possible, accepting anything that looks remotely like a bitmap.

INTERNET MEDIA TYPE

No Internet Media Type (aka MIME type, content type) for PBM has been registered with IANA, but the value `image/x-portable-bitmap` is conventional.

Note that the PNM Internet Media Type `image/x-portable-anymap` also applies.

FILE NAME

There are no requirements on the name of a PBM file, but the convention is to use the suffix ".pbm". "pnm" is also conventional, for cases where distinguishing between the particular subformats of PNM is not convenient.

COMPATIBILITY

Before July 2000, there could be at most one image in a PBM file. As a result, most tools to process PBM files ignore (and don't read) any data after the first image.

NAME

PPM - Netpbm color image format

DESCRIPTION

This program is part of Netpbm.

The PPM format is a lowest common denominator color image file format.

It should be noted that this format is egregiously inefficient. It is highly redundant, while containing a lot of information that the human eye can't even discern. Furthermore, the format allows very little information about the image besides basic color, which means you may have to couple a file in this format with other independent information to get any decent use out of it. However, it is very easy to write and analyze programs to process this format, and that is the point.

It should also be noted that files often conform to this format in every respect except the precise semantics of the sample values. These files are useful because of the way PPM is used as an intermediary format. They are informally called PPM files, but to be absolutely precise, you should indicate the variation from true PPM. For example, "PPM using the red, green, and blue colors that the scanner in question uses."

The name "PPM" is an acronym derived from "Portable Pixel Map." Images in this format (or a precursor of it) were once also called "portable pixmaps."

THE FORMAT

The format definition is as follows. You can use the libnetpbm C subroutine library to read and interpret the format conveniently and accurately.

A PPM file consists of a sequence of one or more PPM images. There are no data, delimiters, or padding before, after, or between images.

Each PPM image consists of the following:

1. A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P6".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
4. Whitespace.
5. A height, again in ASCII decimal.
6. Whitespace.
7. The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero.
8. A single whitespace character (usually a newline).
9. A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

   A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

   In the raster, the sample values are "nonlinear." They are proportional to the intensity of the ITU-R Recommendation BT.709 red, green, and blue in the pixel, adjusted by the BT.709 gamma transfer function. (That transfer function specifies a gamma number of 2.2 and has a linear section for small intensities). A value of Maxval for all three samples represents CIE D65 white and the most intense color in the color universe of which the image is part (the color universe is all the colors in all images to which this image might be compared).

   BT.709's range of channel values (16-240) is irrelevant to PPM.

   ITU-R Recommendation BT.709 is a renaming of the former CCIR Recommendation 709. When CCIR was absorbed into its parent organization, the ITU, ca. 2000, the standard was renamed. This document once referred to the standard as CIE Rec. 709, but it isn't clear now that CIE ever sponsored such a standard.

   Note that another popular color space is the newer sRGB. A common variation from PPM is to substitute this color space for the one specified. You can use **pnmgamma** to convert between this variation and true PPM.

   Note that a common variation from the PPM format is to have the sample values be "linear," i.e. as specified above except without the gamma adjustment. **pnmgamma** takes such a PPM variant as input and produces a true PPM as output.

Strings starting with "#" may be comments, the same as with PBM.

Note that you can use **pamdepth** to convert between a the format with 1 byte per sample and the one with 2 bytes per sample.

All characters referred to herein are encoded in ASCII. "newline" refers to the character known in ASCII as Line Feed or LF. A "white space" character is space, CR, LF, TAB, VT, or FF (I.e. what the ANSI standard C isspace() function calls white space).

Plain PPM

There is actually another version of the PPM format that is fairly rare: "plain" PPM format. The format above, which generally considered the normal one, is known as the "raw" PPM format. See **pbm** for some commentary on how plain and raw formats relate to one another and how to use them.

The difference in the plain format is:

- There is exactly one image in a file.
- The magic number is P3 instead of P6.
- Each sample in the raster is represented as an ASCII decimal number (of arbitrary size).
- Each sample in the raster has white space before and after it. There must be at least one character of white space between any two samples, but there is no maximum. There is no particular separation of one pixel from another -- just the required separation between the blue sample of one pixel from the red sample of the next pixel.
- No line should be longer than 70 characters.

Here is an example of a small image in this format.

```
P3
# feep.ppm
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
```

There is a newline character at the end of each of these lines.

Programs that read this format should be as lenient as possible, accepting anything that looks remotely like a PPM image.

INTERNET MEDIA TYPE

No Internet Media Type (aka MIME type, content type) for PPM has been registered with IANA, but the value `image/x-portable-pixmap` is conventional.

Note that the PNM Internet Media Type `image/x-portable-anymap` also applies.

FILE NAME

There are no requirements on the name of a PPM file, but the convention is to use the suffix ".ppm". "pnm" is also conventional, for cases where distinguishing between the particular subformats of PNM is not convenient.

COMPATIBILITY

Before April 2000, a raw format PPM file could not have a maxval greater than 255. Hence, it could not have more than one byte per sample. Old programs may depend on this.

Before July 2000, there could be at most one image in a PPM file. As a result, most tools to process PPM files ignore (and don't read) any data after the first image.

RAW Image Format

There are two supported file format versions:

- Pure raw binary data only.
- ASCII header (7 lines) followed by the pure raw binary data.

The 7 header lines contain the following keys and values:

`Magic=%s`	`File format identifier. Fixed value „RAW“.`
`Width=%d`	`Image width in pixels.`
`Height=%d`	`Image height in pixels.`
`NumChan=%d`	`Possible values: 1 or 3.`
`ByteOrder=%s`	`Possible values: "Intel" or "Motorola".`
`ScanOrder=%s`	`Possible values: "TopDown" or "BottomUp".`
`PixelType=%s`	`Possible values: "byte", "short", "int", "float" or "double".`

Example 1:

Header for a 32-bit floating point image of size 256x256 pixels, which contains the image data in little-endian (i.e. Intel architecture) format and stores the scanlines from bottom to top.

```
Magic=RAW
Width=256
Height=256
NumChan=1
ByteOrder=Intel
ScanOrder=BottomUp
PixelType=float
```

Example 2:

Header for a 16-bit unsigned integer image of size 512x512 pixels, which contains the image data in little-endian (i.e. Intel architecture) format and stores the scanlines from top to bottom.

```
Magic=RAW
Width=512
Height=512
NumChan=1
ByteOrder=Intel
ScanOrder=TopDown
PixelType=short
```

RAW files can be read using the PAWT extension and displayed using the Img extension.

Alternatively, these images can be read and visualized with MATLAB or GNU Octave using the functions `ReadRawImageFile` and `ShowImage`.

Usage of these functions is as follows:

```
imgFileRGB = 'Chans3-Bits8.raw';
[img, width, height, numChan, pixelSize] = ReadRawImageFile( imgFileRGB, false );
ShowImage( img, imgFileRGB, width, height, numChan, pixelSize );
```

MATLAB version:

```
function [image, w, h, nc, ps] = ReadRawImageFile( imgName, mapFloatToShort )
  fp = fopen( imgName, 'r' );
  if( fp == -1 )
    error( 'Could not open image: %s', imageName );
  end

  % Read the 7 header lines.
  buf = fgetl( fp );
  magic = sscanf( buf, 'Magic=%s' );
  if( ~strcmp( magic, 'RAW' ) )
    fclose( fp );
    error( 'Wrong value for keyword Magic: %s (must be RAW)', magic );
  end

  buf = fgetl( fp );
  width = sscanf( buf, 'Width=%d' );
  if( width <= 0 )
    fclose( fp );
    error( 'Wrong value for keyword Width: %d (must be greater than zero)', width );
  end

  buf = fgetl( fp );
  height = sscanf( buf, 'Height=%d' );
  if( height <= 0 )
    fclose( fp );
    error( 'Wrong value for keyword Height: %d (must be greater than zero)', height );
  end

  buf = fgetl( fp );
  numChan = sscanf( buf, 'NumChan=%d' );
  if( numChan <= 0 || numChan > 4 )
    fclose( fp );
    error( 'Wrong value for keyword NumChan: %d (must be 1,2,3 or 4)', numChan );
  end

  buf = fgetl( fp );
  byteOrder = sscanf( buf, 'ByteOrder=%s' );
  if( ~strcmp( byteOrder, 'Intel' ) && ~strcmp( byteOrder, 'Motorola' ) )
    fclose( fp );
    error( 'Wrong value for keyword ByteOrder: %s (must be Intel or Motorola)', byteOrder );
  end

  buf = fgetl( fp );
  scanOrder = sscanf( buf, 'ScanOrder=%s' );
  if( ~strcmp( scanOrder, 'BottomUp' ) && ~strcmp( scanOrder, 'TopDown' ) )
    fclose( fp );
    error( 'Wrong value for keyword ScanOrder: %s (must be BottomUp or TopDown)', scanOrder );
  end

  buf = fgetl( fp );
  pixelType = sscanf( buf, 'PixelType=%s' );
  if( ~strcmp( pixelType, 'byte' ) &&
      ~strcmp( pixelType, 'short' ) &&
      ~strcmp( pixelType, 'float' ) )
    fclose( fp );
    error( 'Wrong value for keyword pixelType: %s (must be byte, short or float)', pixelType );
  end

  if( strcmp( byteOrder, 'Motorola' ) )
    fclose( fp );
    error( 'ByteOrder Motorola not yet supported.' );
  end

  % Read the data and transfom into image.
  if( numChan == 1 && strcmp( pixelType, 'float' ) )
    % printf( 'Reading 1-channel floating point data\n' );
    try
      data = fread( fp, width*height, 'float=>float32' );

      if( mapFloatToShort )
        % Map data to uint16
        data_max = max( data );
        data_min = min( data );
        data_out_max = 2^16 - 1;
        data_out_min = 0;

        data_out = uint16((data - data_min) * (data_out_max - data_out_min) /
                   (data_max - data_min) + data_out_min);
        img = reshape( data_out, width, height );
      else
        img = reshape( data, width, height );
      end

      img = img';
      if( strcmp( scanOrder, 'BottomUp' ) )
        img = flipud( img );
      end
    catch err
      fclose( fp );
      error( 'Error reading 1-channel floating point data.' );
    end

  elseif( numChan == 1 && strcmp( pixelType, 'short' ) )
    % printf( 'Reading 1-channel unsigned short data\n' );
    try
      data = fread( fp, width*height, 'uint16=>uint16' );
      img = reshape( data, width, height );
      img = img';
      if( strcmp( scanOrder, 'BottomUp' ) )
        img = flipud( img );
      end
    catch err
      fclose( fp );
      error( 'Error reading 1-channel unsigned short data.' );
    end

  elseif( numChan == 3 && strcmp( pixelType, 'byte' ) )
    % printf( 'Reading 3-channel unsigned byte (RGB) data\n' );
    try
      data = fread( fp, width*height*numChan, 'uint8=>uint8' );
      img = permute( reshape( data, numChan, width ,height ), [3,2,1] );
      if( strcmp( scanOrder, 'BottomUp' ) )
        img = flipud( img );
      end
    catch err
      fclose( fp );
      error( 'Error reading 3-channel unsigned byte (RGB) data.' );
    end

  else
    fclose( fp );
    error( 'Unsupported image format: NumChannels: %d PixelType: %s', numChan, pixelType );
  end

  fclose( fp );

  image = img;
  w  = width;
  h  = height;
  nc = numChan;
  if( strcmp( pixelType, 'byte' ) )
      ps = 1;
   elseif( strcmp( pixelType, 'short' ) )
      ps = 2;
   else
```

```
      ps = 4;
   end
end

function ShowImage( img, imgName, width, height, numChan, pixelSize )
  figure( 'name', imgName );
  if( pixelSize == 1 )
    imshow( img );
  else
    imagesc( img );
    colormap( gray );
  end
  title( sprintf( 'Width: %d Height: %d NumChan: %d PixelSize: %d\n',
                  width, height, numChan, pixelSize ) );
end
```

GNU Octave version:

```
function [image, w, h, nc, ps] = ReadRawImageFile( imgName, mapFloatToShort )
  fp = fopen( imgName, 'r' );
  if( fp == -1 )
    error( 'Could not open image: %s', imageName );
  end

  % Read the 7 header lines.
  buf = fgetl( fp );
  magic = sscanf( buf, 'Magic=%s' );
  if( ! strcmp( magic, 'RAW' ) )
    fclose( fp );
    error( 'Wrong value for keyword Magic: %s (must be RAW)', magic );
  end

  buf = fgetl( fp );
  width = sscanf( buf, 'Width=%d' );
  if( width <= 0 )
    fclose( fp );
    error( 'Wrong value for keyword Width: %d (must be greater than zero)', width );
  end

  buf = fgetl( fp );
  height = sscanf( buf, 'Height=%d' );
  if( height <= 0 )
    fclose( fp );
    error( 'Wrong value for keyword Height: %d (must be greater than zero)', height );
  end

  buf = fgetl( fp );
  numChan = sscanf( buf, 'NumChan=%d' );
  if( numChan <= 0 || numChan > 4 )
    fclose( fp );
    error( 'Wrong value for keyword NumChan: %d (must be 1,2,3 or 4)', numChan );
  end

  buf = fgetl( fp );
  byteOrder = sscanf( buf, 'ByteOrder=%s' );
  if( ! strcmp( byteOrder, 'Intel' ) && ! strcmp( byteOrder, 'Motorola' ) )
    fclose( fp );
    error( 'Wrong value for keyword ByteOrder: %s (must be Intel or Motorola)', byteOrder );
  end

  buf = fgetl( fp );
  scanOrder = sscanf( buf, 'ScanOrder=%s' );
  if( ! strcmp( scanOrder, 'BottomUp' ) && ! strcmp( scanOrder, 'TopDown' ) )
    fclose( fp );
    error( 'Wrong value for keyword ScanOrder: %s (must be BottomUp or TopDown)', scanOrder );
  end

  buf = fgetl( fp );
  pixelType = sscanf( buf, 'PixelType=%s' );
  if( ! strcmp( pixelType, 'byte' ) &&
      ! strcmp( pixelType, 'short' ) &&
      ! strcmp( pixelType, 'float' ) )
    fclose( fp );
    error( 'Wrong value for keyword pixelType: %s (must be byte, short or float)', pixelType );
  end

  if( strcmp( byteOrder, 'Motorola' ) )
    fclose( fp );
    error( 'ByteOrder Motorola not yet supported.' );
  end

  % Read the data and transfom into image.
  if( numChan == 1 && strcmp( pixelType, 'float' ) )
    % printf( 'Reading 1-channel floating point data\n' );
    try
    data = fread( fp, width*height, 'float=>float32' );

    if( mapFloatToShort )
      % Map data to uint16
      data_max = max( data );
      data_min = min( data );
      data_out_max = 2^16 - 1;
      data_out_min = 0;

      data_out = uint16((data - data_min) * (data_out_max - data_out_min) /
                  (data_max - data_min) + data_out_min);
      img = reshape( data_out, width, height );
    else
      img = reshape( data, width, height );
    end

    img = img';
    if( strcmp( scanOrder, 'BottomUp' ) )
      img = flipud( img );
    end
  catch err
    fclose( fp );
    error( 'Error reading 1-channel floating point data.' );
  end

elseif( numChan == 1 && strcmp( pixelType, 'short' ) )
  % printf( 'Reading 1-channel unsigned short data\n' );
  try
    data = fread( fp, width*height, 'uint16=>uint16' );
    img = reshape( data, width, height );
    img = img';
    if( strcmp( scanOrder, 'BottomUp' ) )
      img = flipud( img );
    end
  catch err
    fclose( fp );
    error( 'Error reading 1-channel unsigned short data.' );
  end

elseif( numChan == 3 && strcmp( pixelType, 'byte' ) )
  % printf( 'Reading 3-channel unsigned byte (RGB) data\n' );
  try
    data = fread( fp, width*height*numChan, 'uint8=>uint8' );
    img = permute( reshape( data, numChan, width ,height ), [3,2,1] );
    if( strcmp( scanOrder, 'BottomUp' ) )
      img = flipud( img );
    end
  catch err
    fclose( fp );
    error( 'Error reading 3-channel unsigned byte (RGB) data.' );
  end

else
  fclose( fp );
  error( 'Unsupported image format: NumChannels: %d PixelType: %s', numChan, pixelType );
end

fclose( fp );

image = img;
w  = width;
h  = height;
nc = numChan;
if( strcmp( pixelType, 'byte' ) )
    ps = 1;
 elseif( strcmp( pixelType, 'short' ) )
    ps = 2;
 else
    ps = 4;
 end
```

```
endfunction

function ShowImage( img, imgName, width, height, numChan, pixelSize )
  figure( 'name', imgName );
  if( pixelSize == 1 )
    imshow( img );
  else
    imagesc( img );
    colormap( gray );
  end
  title( sprintf( 'Width: %d Height: %d NumChan: %d PixelSize: %d\n',
                  width, height, numChan, pixelSize ) );
endfunction
```