Specification of the PFS File Format version 1.5

August 23, 2026

1 Introduction

This document contains a detailed specification of the pfs file format. PFS file format is intended to store in particular high-dynamic range images, but it is also flexible enough to store additional data, like depth map or motion flow. To learn how PFS format can be useful and why it is different from existing image formats, see a list of Frequently Asked Questions of the PFS tools package. Information in this section should be sufficient to implement pfs compatible reader or writer.

2 Copyright

Copyright (c) 2003 Rafal Mantiuk and Grzegorz Krawczyk

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

3 Change History

- 1.0 (15.12.2004 RM) — Original version
- 1.1 (14.02.2005 RM) — Colon ':' is no longer allowed in the name of a tag; Added a conceptual UML data-model

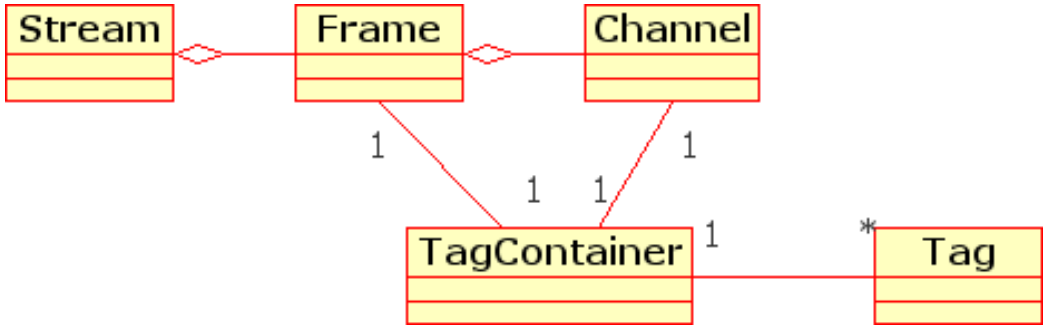


Figure 1: A conceptual UML data model of the pfs-stream

- 1.2 (16.03.2006 RM) — Fixed typos (thanks to grendelkhan)
- 1.3 (16.08.2006 RM) — Added a list of registered channel names; added comments on endianness)
- 1.4 (28.06.2007 RM) — Fixed column-/row-major ambiguity (thanks to Matt)
- 1.5 (06.08.2007 RM) — Specified maximum string lengths and valid value ranges. Added suggestion to prepend custom channel names with "x". (thanks to Martin)
- 1.6 (03.10.2008 RM) — Added a new tag 'BITDEPTH'

4 General Structure

A pfs-stream is a byte-stream that contains one or more frames. Each frame is stored in the pfs-stream right after another frame, without any markers or separators between them. End of file indicates that there are no more frames in the pfs-stream.

A conceptual data model of the pfs stream is shown in Figure 1. A pfs-stream can contain any number of frames, which include any number of channels. Both frame and channel has an associated tag-container, which can contain any number of tags (name=value pairs).

A structure of a single frame is shown in Table 1. A frame encoded in the pfs format consists of a text header followed by binary data. The header contains information on frame size, number of channels and tags. Data items in the header are separated by end of line (EOL) characters. Note that only a unix-type EOL character is allowed (a single character of the ASCII code 10). MSDOS-type EOL is not allowed. To avoid portability problems, use C

string "\x0a" instead of "\n". Note that the header ends with a ENDH string, which is not followed by an EOL character. This prevents C formated IO functions, like fscanf, from reading both the EOL character and first bytes of binary data that happen to have the values of white spaces.

The binary data part of a frame follows text header and holds a 2D array of floating points for each channel. The structure of a channel is described in the next section.

5 Channels

Channels are two-dimensional arrays of floating point numbers, which can contain anything from color data to motion flow and depth (z-buffer) information. Channels are written in a stream in the same order as they are listed in the header. Note that no assumption can be made on the order of channels — it may differ from file to file and some applications may even reverse that order.

For the sake of performance, channel data are encoded in binary format. Each cell (pixel) of an array should be encoded as a IEEE Standard 754 Floating Point Number. Fortunately, this representation is used by CPU for most architectures, including Intel-based PC's, so it is enough to store the values in memory as a C 'float' type and then write them to an IO stream. An array should be encoded in a stream in a row major order — all cells of the first row are followed by the cells of the second row and so on. The encoding starts from the top left corner. The bytes should be encoded in the little-endian order (LSB), which is appropriate for the x86 platform[1].

This version of the pfs format specification defines the following channels:

X — X color component of the CIE XYZ color space. See comments below.

Y — Y color component of the CIE XYZ color space. See comments below.

Z — Z color component of the CIE XYZ color space. See comments below.

DEPTH — Depth channel. Format of the depth information is currently not standardized.

ALPHA — Alpha channel that encodes transparency. The values should be in the range from 0 to 1.

[1]Current implementation of the pfs library does not handle big-endian system, so the pfs files generated on x86 and powerPC platforms are not compatible

Data	Type	Description
PFS1¶		Constant identifier
width height¶	*int int*	Width and height of each channel (1–65535)
channelCount¶	*int*	Number of channels (1–1024)
frameTagCount¶	*int*	Number of tags associated with the frame (0–1024)
for i=1:frameTagCount		
tagName=tagValue¶	*string string*	An i-th tag: name=value pair (max length of `strcat(string, string)` - 1023)
for i=1:channelCount		
channelName¶	*string*	Name of the i-th channel (max string length – 32)
channelTagCount¶	*int*	Number of tags associated with i-th channel (0–1024)
for j=1:channelTagCount		
tagName=tagValue¶	*string string*	An i-th tag: name=value pair (max length of `strcat(string, string)` - 1023)
ENDH		Signalizes the end of the header
for i=1:channelCount		
channelData	*bin-float*[w][h]	Row-major array of 32-bit floating points

Table 1: A structure of pfs-frame. Bold font denotes literal strings. '¶' is a unix-type end of line character (ASCII code 10). Types: *int* – integer value given as string; *string* – a string of one or more characters (1-byte ASCII); *bin-float*[w][h] – a row major array of 32-bit floating point numbers in binary format. The range of valid values and maximum string lengths are given in the parenthesis in the "description" column.

If other information than color, depth and alpha needs to be stored in a channel, a custom name starting with a lower-case "x" letter should be used. If you think that a particular channel name should be registered in this document, feel free to suggest it on the `pfstools@googlegroups.com` discussion group.

Color information must be represented in CIE XYZ (2° standard observer) color space. Depending on the value of the LUMINANCE tag, color data are linear (RELATIVE – linearly related to luminance), linear and calibrated (ABSOLUTE – the values of Y channel represent luminance in cd/m^2), or non-linear (DISPLAY – gamma-corrected). The preferred representation is a 'linear and calibrated'. 'non-linear' is only a temporary representation used before data is written to low dynamic range image files (PNG, JPG) or displayed. For more information refer to the next section — 6. The channels must be named using upper case letters: 'X', 'Y' and 'Z'. For color images, all three X, Y and Z channels must be given. It is enough to include 'Y' channel for gray-level images.

6 Tags

pfs format allows for storing tags, which are 'name=value' pairs. Tags are robust mechanism for specifying additional information within a pfs-stream. Both the name and value should be a text string, i.e. if a floating point number is to be stored, it should be converted into a text string. The name must not contain '=' and ':' character (some programs use notation 'channel_name:tag_name' to assign a tag to the channel). Spaces in front of and after '=' are allowed, although they will not be skipped and they will be interpreted as a part of name or value strings. Tags can be associated with a frame or separate channel, depending how they are placed in the text header of the pfs frame (see Table 1). Each tag associated with a single frame or channel must have an unique name. There is no limit to the number of tags.

In general, tags can be used to store any user defined data, but there are also several tags that are reserved and are part of the pfs format. Those tags let precisely define content of the pfs stream. The reserved tags are listed below:

LUMINANCE specifies a photometric content of the XYZ color channels or Y luminance channel alone. LUMINANCE tag distinguishes between different methods of storing lumiance data. This tag must be associated with a frame and the frame must contain either all XYZ channels or Y channel. Possible values are: ABSOLUTE, RELATIVE, and DISPLAY.

ABSOLUTE Set the LUMINANCE tag to this value if the high-dynamic range data is calibrated. Y channel should contain absolute luminance values given in cd/m^2. Note that luminance values must be ≥ 0 for calibrated content.

RELATIVE High-dynamic range data is of RELATIVE LUMINANCE if it is not calibrated, that is Y channel data is proportional to luminance, but nothing is known about absolute value of luminance. This is the case of most of the high-dynamic range images available on Internet.

DISPLAY If the data comes from low-dynamic range file and no colorimetric information is provided in ICC profile, the data is simply given in pixel values of an unknown display device. This is the worst case, but also most common as most of the low-dynamic range formats (jpeg, png, tiff) do not contain any data that can be used to recover luminance values. The values of the Y channel can range from 0 to 1 and are already gamma corrected.

If the LUMINANCE tag is not specified in the pfs-stream, RELATIVE is assumed.

Example: LUMINANCE=ABSOLUTE

VISUAL_CONTENT tag specifies visual content of the XYZ color channels or Y luminance channel alone. If an image was captured as high-dynamic range photograph, it has one-to-one correspondence with the real word, therefore its VISUAL_CONTENT is MEASUREMENT. However, if the same image has been tone mapped (or gamut mapped) it still resembles appearance of the real word scene, but it has no longer one-to-one correspondence. In this case its VISUAL_CONTENT is RENDERING. This tag is typically set to rendering after tone mapping of high-dynamic range images. This tag must be associated with a frame and the frame must contain either all XYZ channels or Y channel.

MEASUREMENT Set VISUAL_CONTENT tag to MEASUREMENT if the image has one-to-one correspondence with the real word, i.e. it contains measured data.

RENDERING Set VISUAL_CONTENT to RENDERING if the image has the same appearance as the real world scene, but there is no photometric correspondence.

If the tag is not specified in the pfs-stream, MEASUREMENT is assumed. If LUMINANCE tag is set to DISPLAY, the VISUAL_CONTENT

is assumed to be RENDERING.

Example: VISUAL_CONTENT=RENDERING

WHITE_Y defines the value of channel Y that is perceived as reference white. The value should be given in units of an Y channel. If LUMINANCE=ABSOLUTE, the white point is given in cd/m^2. The value of this tag should be a single floating point number in a text format.

Example: WHITE_Y=100

WHITE_x, WHITE_y tags define relative coordinates of color that is perceived as reference white. The value of those tags must be a floating point number in a text format. The value is given as a relative colorimatric coordinates of the CIE XYZ space (2° standard observer) $y = Y/(X + Y + Z)$ and $x = X/(X + Y + Z)$.

If the tag is not specified, D65 white point is assumed: $x = 0.3127\, y = 0.3290$

Example: WHITE_x=0.3457 WHITE_y=0.3585

FILE_NAME contains a string with the name of a source file. This can be used to locate the origin of a frame.

Example: FILE_NAME=my_sequence_frame_0000.hdr

BITDEPTH specifies the number of bits, that are required to store a single channel in a display-referred image (LUMINANCE=DISPLAY) without quality loss. The number should be from 1 to 32.

Example: BITDEPTH=16

7 Discussion

The structure of pfs files is a compromise between simplicity and robustness and (as well) between portability and performance. Therefore a small header is stored in text format as this is the easiest to decode on different platforms. Storing image data as text would result in too much overhead and thus binary format is used. Since parsing a file is typically more difficult to implement than writing it, pfs format tries to make parsing files as easy as possible. This is why sizes of all variable length structures are given first and then actual data follows.