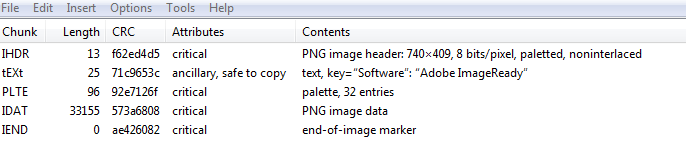
Bug964: "Colour index (n) to palette (n) too big."

# Meaning of the issue being raised

* Assume **Colour Index** is (**byteVal**)
* Assume **Palette** is (**numOfColoursInPalette**)
* Crudely the **Palette** represents a limited number of distinct colours. The **Colour Index** is an index into the Palette table, and represents a pixel.
* The issue here is that it is checking that the colour index that represents a pixel available in the numOfColousInPalette table. In this case it is out of bounds.

# Sample Case

* File name: **04\_13.png**
* TweakPNG information for file:

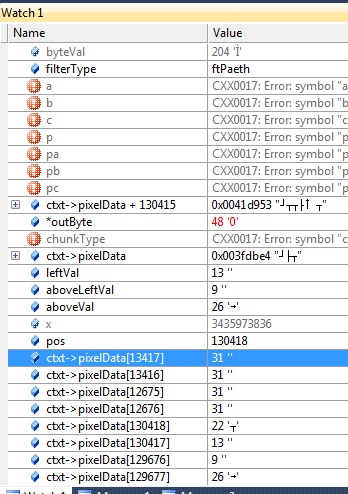


**Deducing the value of numOfColoursInPalette**

* Number of max number of palette entries is based on the bit depth. In this case 2^8 = 256.
* The actual number of palettes is deduced from the PLTE chunk length. In this case **numOfColoursInPalette = 32** (96 / 3 that is PLTE.length/RGB)**.**

**Deducing the value of byteVal**

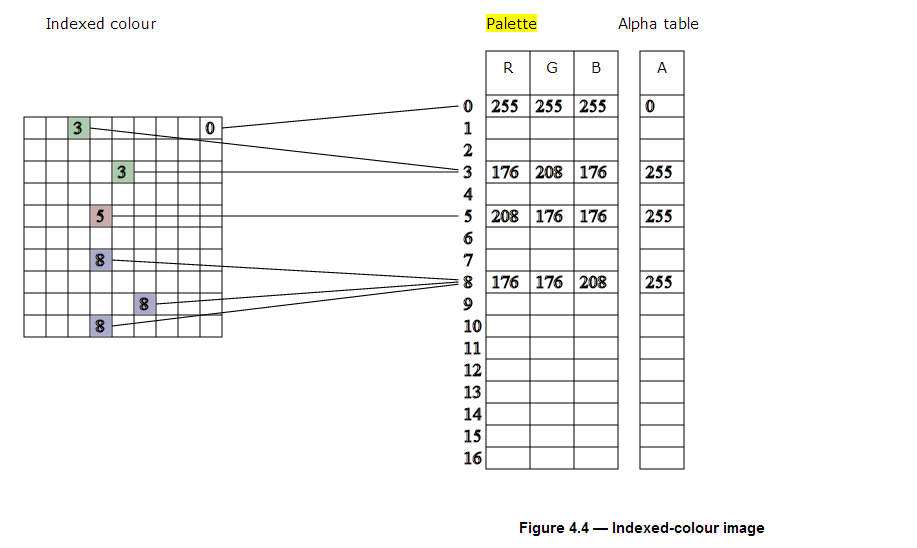
* Encounters an IDAT chunk.
* Zlib inflate occurs on the IDAT chunk and is stored in PNG context ctxt->pixelData
* ctxt->pixelData is interrogated as a number of filter lines (filter types could have been applied to the filter line).
* For each filter line, the 0th index represents the filter type, and the rest of the line is the pixel data.
* Each filter line is 740 bytes.
* In our case, the particular filter line we are looking at has filter type **Paeth** applied. **byteVal** (outByte 48 in debugger) represents the value of the pixel after Paeth reconstruction.
  + The reconstruction formula being
    - Recon(x) = Filt(x) + PaethPredictor(Recon(a), Recon(b), Recon(c))
  + x = ctxt->pixelData[130418] + PaethPredictor(ctxt->pixelData[130417], ctxt->pixelData[129677], ctxt->pixelData[129676])
  + x = 22 + PaethPredictor(13, 26, 9)
  + x = 22 + 26
  + x= 48
  + The current pixel byte being examined ctxt->pixelData is updated with the reconstructed value x.



Appendix A References from http://www.w3.org/

4.3.3 Indexing

If the number of distinct pixel values is 256 or less, and the RGB sample depths are not greater than 8, and the alpha channel is absent or exactly 8 bits deep or every pixel is either fully transparent or fully opaque, then an alternative representation called indexed-colour may be more efficient for encoding. Each pixel is replaced by an index into a palette. The palette is a list of entries each containing three 8-bit samples (red, green, blue). If an alpha channel is present, there is also a parallel table of 8-bit alpha samples.



**Figure 4.4 — Indexed-colour image**

A suggested palette or palettes may be constructed even when the PNG image is not indexed-colour in order to assist viewers that are capable of displaying only a limited number of colours.

For indexed-colour images, encoders can rearrange the palette so that the table entries with the maximum alpha value are grouped at the end. In this case the table can be encoded in a shortened form that does not include these entries.

4.5.1 Introduction

A conceptual model of the process of encoding a PNG image is given in figure 4.7. The steps refer to the operations on the array of pixels or indices in the PNG image. The palette and alpha table are not encoded in this way.

Pass extraction: to allow for progressive display, the PNG image pixels can be rearranged to form several smaller images called reduced images or passes.

Scanline serialization: the image is serialized a scanline at a time. Pixels are ordered left to right in a scanline and scanlines are ordered top to bottom.

Filtering: each scanline is transformed into a filtered scanline using one of the defined filter types to prepare the scanline for image compression.

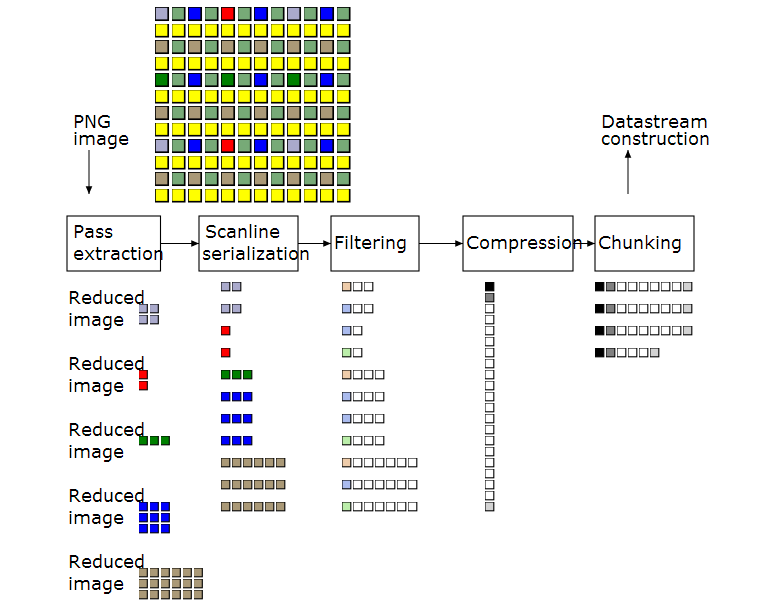
Compression: occurs on all the filtered scanlines in the image.

Chunking: the compressed image is divided into conveniently sized chunks. An error detection code is added to each chunk.

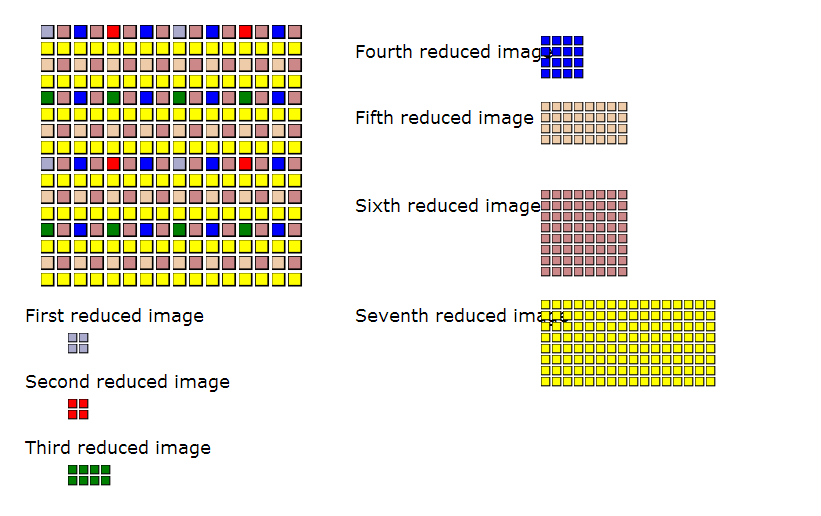
Datastream construction: the chunks are inserted into the datastream.

4.5.2 Pass extraction

Pass extraction (see figure 4.8) splits a PNG image into a sequence of reduced images where the first image defines a coarse view and subsequent images enhance this coarse view until the last image completes the PNG image. The set of reduced images is also called an interlaced PNG image. Two interlace methods are defined in this International Standard. The first method is a null method; pixels are stored sequentially from left to right and scanlines from top to bottom. The second method makes multiple scans over the image to produce a sequence of seven reduced images. The seven passes for a sample image are illustrated in figure 4.8. See clause 8: Interlacing and pass extraction.



**Figure 4.7 — Encoding the PNG image**



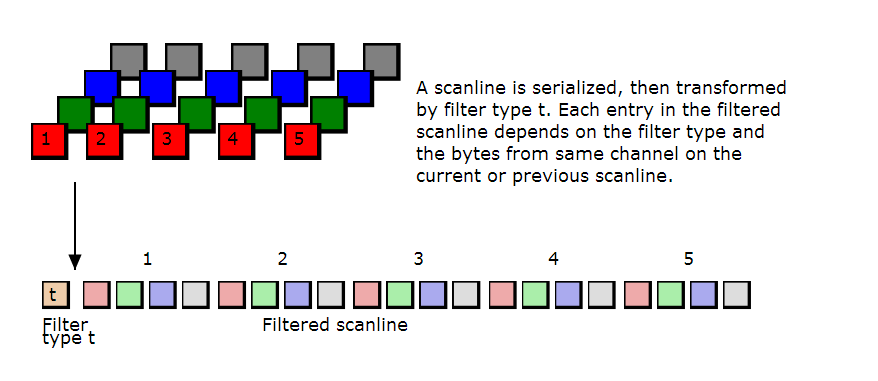
**Figure 4.8 — Pass extraction**

4.5.3 Scanline serialization

Each row of pixels, called a scanline, is represented as a sequence of bytes.

4.5.4 Filtering

PNG standardizes one filter method and several filter types that may be used to prepare image data for compression. It transforms the byte sequence in a scanline to an equal length sequence of bytes preceded by a filter type byte (see figure 4.9 for an example). The filter type byte defines the specific filtering to be applied to a specific scanline. The encoder shall use only a single filter method for an interlaced PNG image, but may use different filter types for each scanline in a reduced image. See clause 9: Filtering.



**Figure 4.9 — Serializing and filtering a scanline**

9.2 Filter types for filter method 0

Filters are applied to **bytes**, not to pixels, regardless of the bit depth or colour type of the image. The filters operate on the byte sequence formed by a scanline that has been represented as described in 7.2: Scanlines. If the image includes an alpha channel, the alpha data is filtered in the same way as the image data.

Filters may use the original values of the following bytes to generate the new byte value:

|  |  |
| --- | --- |
| x | the byte being filtered; |
| a | the byte corresponding to x in the pixel immediately before the pixel containing x (or the byte immediately before x, when the bit depth is less than 8); |
| b | the byte corresponding to x in the previous scanline; |
| c | the byte corresponding to b in the pixel immediately before the pixel containing b (or the byte immediately before b, when the bit depth is less than 8). |

Figure 9.1 shows the relative positions of the bytes x, a, b, and c.

PNG filter method 0 defines five basic filter types as listed in Table 9.1. Orig(y) denotes the orginal (unfiltered) value of byte y. Filt(y) denotes the value after a filter has been applied.Recon(y) denotes the value after the corresponding reconstruction function has been applied. The filter function for the Paeth type PaethPredictor is defined below.

Filter method 0 specifies exactly this set of five filter types and this shall not be extended. This ensures that decoders need not decompress the data to determine whether it contains unsupported filter types: it is sufficient to check the filter method in **IHDR**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table 9.1 — Filter types** | | | |
| **Type** | **Name** | **Filter Function** | **Reconstruction Function** |
| 0 | None | Filt(x) = Orig(x) | Recon(x) = Filt(x) |
| 1 | Sub | Filt(x) = Orig(x) - Orig(a) | Recon(x) = Filt(x) + Recon(a) |
| 2 | Up | Filt(x) = Orig(x) - Orig(b) | Recon(x) = Filt(x) + Recon(b) |
| 3 | Average | Filt(x) = Orig(x) - floor((Orig(a) + Orig(b)) / 2) | Recon(x) = Filt(x) + floor((Recon(a) + Recon(b)) / 2) |
| 4 | Paeth | Filt(x) = Orig(x) - PaethPredictor(Orig(a), Orig(b), Orig(c)) | Recon(x) = Filt(x) + PaethPredictor(Recon(a), Recon(b), Recon(c)) |

For all filters, the bytes "to the left of" the first pixel in a scanline shall be treated as being zero. For filters that refer to the prior scanline, the entire prior scanline and bytes "to the left of" the first pixel in the prior scanline shall be treated as being zeroes for the first scanline of a reduced image.

To reverse the effect of a filter requires the decoded values of the prior pixel on the same scanline, the pixel immediately above the current pixel on the prior scanline, and the pixel just to the left of the pixel above.

Unsigned arithmetic modulo 256 is used, so that both the inputs and outputs fit into bytes. Filters are applied to each byte regardless of bit depth. The sequence of Filt values is transmitted as the filtered scanline.

9.4 Filter type 4: Paeth

The Paeth filter function computes a simple linear function of the three neighbouring pixels (left, above, upper left), then chooses as predictor the neighbouring pixel closest to the computed value. The algorithm used in this International Standard is an adaptation of the technique due to Alan W. Paeth [PAETH].

The PaethPredictor function is defined in the code below. The logic of the function and the locations of the bytes a, b, c, and x are shown in figure 9.1. Pr is the predictor for byte x.

p = a + b - c

pa = abs(p - a)

pb = abs(p - b)

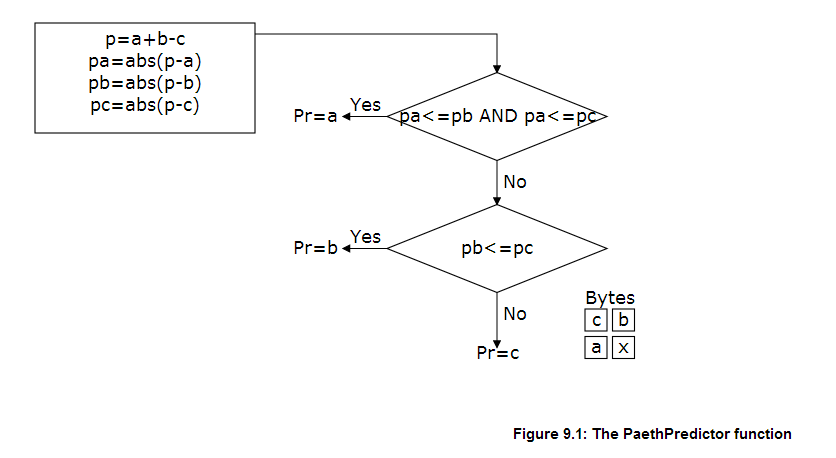
pc = abs(p - c)

if pa <= pb and pa <= pc then Pr = a

else if pb <= pc then Pr = b

else Pr = c

return Pr



**Figure 9.1: The PaethPredictor function**

The calculations within the PaethPredictor function shall be performed exactly, without overflow.

**The order in which the comparisons are performed is critical and shall not be altered.** The function tries to establish in which of the three directions (vertical, horizontal, or diagonal) the gradient of the image is smallest.

Exactly the same PaethPredictor function is used by both encoder and decoder.

11.2.3 PLTE Palette

The four-byte chunk type field contains the decimal values

80 76 84 69

The **PLTE** chunk contains from 1 to 256 palette entries, each a three-byte series of the form:

|  |  |
| --- | --- |
| Red | 1 byte |
| Green | 1 byte |
| Blue | 1 byte |

The number of entries is determined from the chunk length. A chunk length not divisible by 3 is an error.

This chunk shall appear for colour type 3, and may appear for colour types 2 and 6; it shall not appear for colour types 0 and 4. There shall not be more than one **PLTE** chunk.

For colour type 3 (indexed-colour), the **PLTE** chunk is required. The first entry in **PLTE** is referenced by pixel value 0, the second by pixel value 1, etc. The number of palette entries shall not exceed the range that can be represented in the image bit depth (for example, 24 = 16 for a bit depth of 4). It is permissible to have fewer entries than the bit depth would allow. In that case, any out-of-range pixel value found in the image data is an error.

For colour types 2 and 6 (truecolour and truecolour with alpha), the **PLTE** chunk is optional. If present, it provides a suggested set of colours (from 1 to 256) to which the truecolour image can be quantized if it cannot be displayed directly. It is, however, recommended that the **sPLT** chunk be used for this purpose, rather than the **PLTE** chunk. If neither **PLTE** nor **sPLT** chunks are present and the image cannot be displayed directly, quantization has to be done by the viewing system. However, it is often preferable for the selection of colours to be done once by the PNG encoder. (See 12.6: Suggested palettes.)

Note that the palette uses 8 bits (1 byte) per sample regardless of the image bit depth. In particular, the palette is 8 bits deep even when it is a suggested quantization of a 16-bit truecolour image.

There is no requirement that the palette entries all be used by the image, nor that they all be different.

Appendix B References from http://www.libpng.org/

8.5.1. Palette-Based

Palette-based images, also known as colormapped or index-color images, use the PLTE chunk and are supported in four pixel depths: 1, 2, 4, and 8 bits, corresponding to a maximum of 2, 4, 16, or 256 palette entries. Unlike GIF images, however, fewer than the maximum number of entries may be present. On the other hand, GIF does support pixel depths of 3, 5, 6, and 7 bits; 6-bit (64-color) images, in particular, are common on the World Wide Web.

TIFF also supports palette images, but baseline TIFF allows only 4- and 8-bit pixel depths. Perhaps a more useful comparison is with the superset of baseline TIFF that is supported by Sam Leffler's free libtiff, which has become the software industry's unofficial standard for TIFF decoding. libtiff supports palette bit depths of 1, 2, 4, 8, and 16 bits. Unlike PNG and GIF, however, the TIFF palette always uses 16-bit integers for each red, green, and blue value, and as with GIF, all 2bit depth entries must be present in the file. Nor is there any provision for compression of the palette data--so a 16-bit TIFF palette would require 384 KB all by itself.

Appendix C References from http://en.wikipedia.org/wiki/Indexed\_color

Palette size[edit]

*Main article: Palette (computing)*

The palette itself stores a limited number of distinct colors; 4, 16 or 256 are the most common cases. These limits are often imposed by the target architecture's display adapterhardware, so it is not a coincidence that those numbers are exact powers of two (the binary code): 22 = 4, 24 = 16 and 28 = 256. While 256 values can be fit into a single 8-bit byte(and then a single indexed color pixel also occupies a single byte), pixel indices with 16 (4-bit, a nibble) or fewer colors can be packed together into a single byte (two nibbles per byte, if 16 colors are employed, or four 2-bit pixels per byte if using 4 colors). Sometimes, 1-bit (2-color) values can be used, and then up to eight pixels can be packed into a single byte; such images are considered *binary images* (sometimes referred as a *bitmap* or *bilevel image*) and not an indexed color image.

video overlay is intended through a *transparent color*, one palette entry is specifically reserved for this purpose, and it is discounted as an available color. Some machines, such as the MSX series, had the transparent color reserved by hardware.[5]

Indexed color images with palette sizes beyond 256 entries are rare. The practical limit is around 12-bit per pixel, 4,096 different indices. To use indexed 16 bpp or more does not provide the benefits of the indexed color images' nature, due to the color palette size in bytes being greater than the raw image data itself. Also, useful direct RGB Highcolor modes can be used from 15 bpp and up.

If an image has many subtle color shades, it is necessary to select a limited repertoire of colors to approximate the image using color quantization. Such a palette is frequently insufficient to represent the image accurately; difficult-to-reproduce features such as gradients will appear blocky or as strips (*color banding*). In those cases, it is usual to employ dithering, which mixes different-colored pixels in patterns, exploiting the tendency of human vision to blur nearby pixels together, giving a result visually closer to the original one.