PNG Development Group
png-list@dworkin.wustl.edu
30 Mar 1996

PNG Proposed Chunks, draft 0.960330

File

png-proposed-chunks-0.960330.txt

Status of this Memo

This document is an informal draft of the PNG development group.

Comments on this document can be sent to the PNG specification maintainers at `png-info@uunet.uu.net` or at `png-list@dworkin.wustl.edu`.

Distribution of this memo is unlimited.

At present, the latest version of this document is available on the World Wide Web from `ftp://swrinde.nde.swri.edu/pub/png-group/documents/`.

Notices

Abstract

This document describes some special-purpose chunk types that have been proposed for use in various PNG (Portable Network Graphics) applications. They have not yet been approved for registration by the PNG developers, and therefore are experimental and their format is subject to change. The proposed chunks are `alIG`, `drNG/DrNG`, `faLS`, `fiNG`, `loGE/LoGE`, `spLT`, `xySC`, and `zsCL`.

Contents

1 Introduction

Chunks described here have been proposed to the PNG development group. Their definitions are subject to change until such time as the group formally registers them. No decoder is required or expected to implement these chunks except for experimental or evaluation purposes. Comments on these proposals, and new proposals for additional chunk types, should be sent to the PNG specification maintainers at `png-info@uunet.uu.net`. The basic PNG specification is available from the W3C archive at `http://www.w3.org/pub/WWW/TR/WD-png`.

2 Using Proposed PNG Chunks

No chunks described in this document have yet been registered by the PNG maintainers. Therefore they have a lower-case letter in the second position of the chunkname. They are **experimental** chunks and the format is subject to change. If and when any become registered, the second letter of the chunk name will become uppercase, the chunk description will be moved either to a PNG extensions document or to the core PNG specification, and there will be no further change to the format.

Whenever you use any of these unregistered chunks you should also include a `tEXt` chunk describing it, for example:

```
tEXtComment\0
This file contains a zsCL chunk written according to the
format given in Version 0.960330 of the PNG Proposed Chunks document.
```

3 Summary of Proposed Chunks

This table summarizes some properties of the proposed chunks described in this document.

Chunk name	Multiple OK?	Ordering constraints
alIG	No	Before IDAT
drNG/DrNG	No	Before IDAT
faLS	No	Before IDAT
fiNG	No	None
loGE/LoGE	No	Before IDAT
spLT	Yes	Before IDAT
xySC	No	Before IDAT
zsCL	No	Before IDAT

4 Proposed Chunk Specifications

This section provides the detailed specifications for the proposed chunks.

4.1 `alIG` Alignment

This chunk may be used to provide alignment information to applications. It is especially useful for transparent images in which the edges of the interesting part of the picture do not correspond exactly to the edges of the image data. It can be helpful in aligning the picture with adjacent text, if there is a specific location in the picture that should be aligned with the text baseline. (We use "picture" to mean the thing depicted, as opposed to "image" which is the entire width x height rectangle described by the PNG stream.)

This chunk's contents are

```
left     4 bytes  left side of picture
center   4 bytes  center of picture (horizontal)
right    4 bytes  right side of picture
top      4 bytes  top of picture
middle   4 bytes  middle of picture (vertical)
baseline 4 bytes  typographic baseline of picture (vertical)
bottom   4 bytes  bottom of picture
```

All values are signed 32-bit integer values, measured in pixels downward from the top of the image or rightward from the left edge of the image.

Applications will not normally try to use all of the alignment values at once. A browser might use the "center" value when it wishes to center the image left-to-right on a page. A typographic system might use the "baseline" value to line up an image that contains a fancy capital letter with the baseline of the adjacent text, or the "middle" value to line up images containing mathematical equations. Typographic systems could also determine the typographic "height", "depth", "width", and reference point values:

```
font_height = baseline - top
font_width  = right - left
font_depth  = bottom - baseline
font_ref_x  = left
font_ref_y  = baseline
```

If the `alIG` chunk is not present, applications can assume the following values:

```
left     = 0
center   = width/2
right    = width
top      = 0
middle   = height/2
baseline = 3/4 * height
bottom   = height
```

Only one `alIG` chunk is permitted in a PNG datastream. If present, it must appear prior to the first `IDAT` chunk.

4.2 `drNG/DrNG` Pixel display range

This chunk may be used when the pixel values do not occupy the full range of possible values, and when bit depth scaling is not appropriate. It can also be used to enhance contrast by scaling to a larger range than the actual range of pixel values. It can be used to correct the brightness of an image that has been scaled by zero-filling rather than linear scaling. See [link to sBIT in core spec]. The chunk defines the pixel values that correspond to minimum and maximum intensity, to which the pixel data is to be scaled.

The chunk can be used as a critical chunk, named `DrNG`, or as an ancillary chunk, named `drNG`. The syntax and function is exactly the same whichever name is used. The critical version should only be used if the image cannot be meaningfully displayed without performing `DrNG` scaling. Decoders not recognizing `DrNG` will not attempt to display the image at all. Encoders are strongly encouraged to scale the image data properly and to use the noncritical version, `drNG`, if at all possible.

This chunk's contents are

```
min_sample  m bytes   Raw sample value corresponding to
                      minimum intensity of the graylevel
                      or red channel, written as a text
                      floating-point value [link to
                      Floating-Point Values in extensions
                      document]

0            1 byte   a zero byte to separate fields

max_sample  n bytes   Raw sample value corresponding to
                      maximum intensity for the graylevel
                      or red channel

(The following entries are omitted for color_types 0 and 4)

0           1 byte    a zero byte to separate fields

min_green  m bytes    raw sample value corresponding to
                      minimum green intensity

0           1 byte    a zero byte to separate fields

max_green  m bytes    raw sample value corresponding to
                      maximum green intensity

0           1 byte    a zero byte to separate fields

min_blue   m bytes    raw sample value corresponding to
                      minimum blue intensity

0           1 byte    a zero byte to separate fields

max_blue   m bytes    raw sample value corresponding to
                      maximum blue intensity
```

If this chunk is present, graylevel or color sample values are to be scaled for display between minimum and maximum intensity by linear interpolation. When the sample value falls outside the range min_value..max_value, it is to be set to min_value or max_value as appropriate. For each graylevel or color sample, the conversion is

```
ratio := (2^bit_depth - 1)/(max_value - min_value)
result := (input_sample - min_value) * ratio
output_sample := LIMIT (0, result, 2^bit_depth-1)

in which the function LIMIT (low, x, high) is
MAX (MIN (x, high), low).
```

Note that min_value and max_value are permitted to be negative, positive, or zero. The only restriction is that they be different from each other.

Encoders **should not** use `drNG/DrNG` in lieu of reasonably scaling the samples. For example, if the sample values range from 0 (black) to 1000 (white), it would be an extremely bad idea to use the `drNG` chunk to request the decoder to scale 1000 up to 65535, because a PNG viewer that does not understand `drNG/DrNG` (and, since this is an extension chunk, most viewers will probably not understand `drNG/DrNG`) would simply display a very dark rectangle. Instead, multiply your samples by 64, and use `drNG` with max_sample=64000 to request the decoder to do the final adjustment.

If the `tRNS` chunk is present, its value is compared to the unscaled pixel value, prior to applying the `drNG/DrNG` scaling. The `faLS`, `gAMA`, `cHRM`, and alpha conversions, if present, are applied to the scaled sample values.

If present, this chunk must appear before the first `IDAT` chunk. Only one `drNG/DrNG` chunk is permitted in a PNG file.

4.3 `faLS` False-color Palette

This chunk is used with grayscale data, where the pixels are to be rendered in a false color according to the grayscale value. The grayscale sample of each pixel serves as a pointer into the false-color palette.

This chunk contains from 1 to 2^bitdepth 8-byte false-color palette entries:

```
index:          2 bytes, range 0 .. (2^bitdepth) -1)
red:            2 bytes, range 0 .. 65535
green:          2 bytes, range 0 .. 65535
blue:           2 bytes, range 0 .. 65535

etc.
```

The number of entries is determined from the chunk length. This length not divisible by 8 is an error.

This chunk may appear for color type 0 or color type 4. If it appears for any other color type, it will be ignored.

The complete 2^bitdepth-entry false-color palette can be built from the chunk data. If the first entry (index value 0) is missing, it will be assumed to be {0, 0, 0} (black). If the last entry (index value 2^bitdepth - 1) is missing, it will be assumed to be {65535, 65535, 65535} (white). The red, green, blue samples for other missing entries are filled in by linearly interpolating between the samples that are present, independently for each of the three color components.

Once the complete false-color palette is established, it is used similarly to PLTE. The first entry in the completed false-color palette is referenced by the grayscale value 0, the second by grayscale value 1, etc.

If the `tRNS` chunk is present, its value is compared to the graylevel, not to the converted false-color of the pixels. If the `bKGD` chunk is present, background pixels will be displayed in the false-color corresponding to the grayscale value found in the `bKGD` chunk. The `gAMA`, `cHRM`, and alpha conversions, if present, are applied to the color samples in the converted false-color pixels.

Note that the `gAMA` and other values must be selected so that the grayscale image is correctly displayed when the `faLS` chunk is unrecognized or ignored.

If this chunk does appear, it must precede the first `IDAT` chunk. There may be only one `faLS` chunk in a PNG file.

4.4 `fiNG` Fingerprint

This chunk contains an ADLER-32 fingerprint of the uncompressed, unfiltered, uninterlaced image data, that can be used for rapidly determining whether two files contain the same image, without having to actually decode the IDAT chunks.

The `fiNG` chunk's contents are

```
fingerprint 4 bytes
```

The fingerprint is the ADLER-32 check value computed on the image data, promoted by left-bit-replication (see bit_depth scaling in PNG spec) to 16-bit RGBA (colortype 6). Scaling is done without regard to the `sBIT` chunk, if present. It is calculated without regard to any ancillary chunks. The value is computed on the data in the same order as the bytes would appear in a color type 6 datastream, except that no filter bytes are included in the calculation. The ADLER-32 check value is computed as described in [link to adler-32 code in zlib docs]

An image's fingerprint will not change if the image is changed from interlaced to noninterlaced, or compressed with a different method, or filtered differently, or if any ancillary chunks are added, modified, or removed. It will change if there is any lossy conversion of the pixel data or if the PLTE data is changed in an indexed-color image.

While the `fiNG` chunk does not have any ordering requirements, You should put it early in the file for quick access.

4.5 `loGE/LoGE` Logarithmic Encoding

The `loGE/LoGE` chunk is used to decode image data whose grayscale or color samples have been logarithmically encoded.

The chunk can be used as a critical chunk, named `LoGE`, or as an ancillary chunk, named `loGE`. The syntax and function is exactly the same whichever name is used. The critical version should only be used if the image cannot be meaningfully displayed without performing `LoGE` scaling. Decoders not recognizing `LoGE` will not attempt to display the image at all. Encoders are strongly encouraged to include a `gAMA` chunk that permits a meaningful if not completely accurate display, and use the noncritical version, `loGE`, if at all possible.

The `loGE/LoGE` chunk's contents are

```
P0             n0 bytes    (First parameter, a real number
                           written as a text floating-point
                           value [link to Floating-Point
                           Values in extensions document]
Null separator: 1 byte

P1             n1 bytes    (Second parameter)

Null separator: 1 byte

P2             n2 bytes    (Third parameter)
```

There is no trailing zero for the final string.

The scaling algorithm is

```
normalized_sample_value := input_sample/(2^input_bitdepth-1)
scaled_value := P0 + P1 * P2^normalized_sample_value
output_sample := LIMIT (0, scaled_value, 2^output_bitdepth-1)

in which the function LIMIT (low, x, high) is
MAX (MIN (x, high), low).
```

For color_types 0 and 4, the normalized sample value is the grayscale value of the pixel normalized to the range [0.0:1.0] by dividing it by the maximum value for the input bit depth, using floating point arithmetic.

For color_types 2, 3, and 6, the scaling algorithm is applied independently to each of the color samples similarly normalized to [0.0:1.0].

The alpha channel, if present, is not affected by the `loGE/LoGE` transformation. Alpha compositing is done in the normal manner, with the transformed pixels forming the foreground image [link to Decoders: Alpha Processing in core spec]

Pure logarithmic data can be expressed with

```
P0 = 0
P1 = min_val
P2 = max_val/min_val
```

In which the range [min_val..max_val] includes the minimum and maximum values appearing in the source data.

When the image is decoded using the `loGE/LoGE` data, the gamma calculation for the decoded samples should be done as though the file_gamma were 1.0, regardless of the contents of the `gAMA` chunk.

It is advisable to include a `gAMA` chunk with logarithmic data, in case a viewer does not use the `loGE` data to decode it. A value of gamma should be chosen that allows the image to be displayed as well as it can be with a viewer that supports `gAMA` but not `loGE`.

You can select a value of gamma (also called file_gamma) by eye as described in [link to Gamma Tutorial: General Gamma Handling in core spec], or you can calculate one as follows:

If the maximum value of the logarithmically decoded pixels can reasonably be displayed as white on the monitor, then specifying a value of gamma given by

```
gamma = LN( LN(0.2) / LN(max_val/min_val) + 1) / LN(0.2)
(in which LN() is the natural logarithm function)
```

causes the two values max_val and 0.2*max_val to be reproduced at correct brightness on screen - these can be thought of as white and mid-grey. Tones between white and mid-grey will be a bit too bright, ones darker than mid-grey will be increasingly too dark, but it's a reasonable approximation for viewable display. A max_val/min_val ratio of 64 gives a gamma of about 0.3, a max_val/min_val of 1000 gives a gamma of about 0.165, so a viewer assuming a default gamma of 0.45 is going to give a bright, washed-out image for any log-encoded image.

In cases where the maximum sample value is many times brighter than scene white (i.e. the image is encoded to retain specular highlight information), the formula above doesn't apply, but there's probably no way of getting such an image to display reasonably anyway with a viewer that doesn't understand `loGE`.

If present, the `loGE/LoGE` chunk must appear before the first `IDAT` chunk. Only one instance of the `loGE/LoGE` chunk is permitted in a PNG datastream.

4.6 `spLT` Suggested Palette

This chunk may be used to suggest a reduced palette to be used when the display device is not capable of displaying the full range of colors present in the image. If present, it provides a recommended set of colors, with alpha and frequency information, that may be used to construct a reduced palette to which the truecolor image may be quantized.

This chunk's contents are a zero-byte-terminated text string that names the palette, followed by a series of palette entries, each a ten-byte series, containing five unsigned integers:

```
name:      n bytes (ASCII text)
null byte  1 byte  (terminator)
```

```
red:       2 bytes (0 = black, 65535 = red)
green:     2 bytes (0 = black, 65535 = green)
blue:      2 bytes (0 = black, 65535 = blue)
alpha:     2 bytes (0 = fully transparent, 65535 = fully opaque)
frequency: 2 bytes (relative frequency of occurrence)
...
```

There can be any number of entries; a decoder determines the number of entries from the remaining chunk length after the null-terminated "name" string. This length not divisible by ten is an error. Entries must appear in decreasing order of "frequency".

The "name" (e.g. "rgba512 8-8-4-2 color cube", "rgb256 winter scenery", "rgb242 6-6-6 color cube plus 26 gray levels", "50-color rgb palette for use with early versions of Mosaic") identifies the palette, which may permit applications or people to choose the appropriate one when more than one suggested palette appears in a PNG file. The "name" string must consist only of printable ASCII characters and may not have leading or trailing blanks, but may have single embedded blanks. There must be at least one and no more than 79 characters in the name. Names are case-sensitive.

The red, green, and blue values are not premultiplied by alpha, nor are they precomposited against any background. A decoder can build a palette by compositing those palette entries against any background color or set of background colors that it chooses. See [link to bKGD]

Each frequency entry is proportional to the fraction of pixels in the image that are closest to that palette entry, without regard to any compositing against a background palette. The exact scale factor is chosen by the encoder, but should be chosen so that the range of individual values reasonably fills the range 0 to 65535. It is acceptable to artificially inflate the "frequency" values for "important" colors such as those in a company logo or in the facial features of a portrait. Zero is a valid value for frequency.

Note that the palette uses 16 bits (2 bytes) per value regardless of the image bit depth specification. Decoders wishing to construct 8-bit palettes can accomplish this by scaling down the RGB entries to 8 bits. [link to bit_depth rescaling].

Note: Earlier versions of the PNG specification recommended that the `PLTE` [link to PLTE] and `hIST` chunks be used for this purpose. While this is still allowed, to maintain backward compatibility, the `spLT` chunk is preferable, particularly when transparent pixels are present. When both the `PLTE` and `spLT` chunks are present, the `PLTE` data should only be used for decoding the indexed-color (color type 3) pixels, and the `spLT` data should be used for constructing the display palette. If the `hIST` chunk is also present, decoders that process the `spLT` chunk should ignore it.

This chunk may appear for any color type. If this chunk does appear, it must precede the first `IDAT` chunk. There may be multiple `spLT` chunks, with different names.

4.7 xySC X and Y-Scale of Image Subject

This chunk relates the actual dimension of the image subject in the "x" (width) and "y" (height) direction to the column and row position of a pixel. It provides some additional capability beyond that available in the `sCAL` [link to sCAL in PNG extensions document] chunk, which does not provide for an offset, and does not provide for different units in the width and height directions. The latter could be useful, for example, in racing photo-finish "streak" images, where one of the dimensions is time and the other is length. The chunk does not affect the image display, but could be used to construct scaled axes adjacent to the image.

The `xySC` chunk's contents are

```
Xunit          xu bytes  (Latin-1 symbol or description of
                         the width unit, eg. milliseconds,
                         degrees West Longitude,  etc).
                         A zero-length string may be used if
                         the data is dimensionless.

Null separator: 1 byte

Xoffset        xo bytes  X offset, a real number written as a
                         text floating-point value [link to
                         Floating-Point Values in extensions
                         document] This is the physical value
                         of "x" corresponding to the left
                         edge of the image.

Null separator: 1 byte

Xscale         xs bytes  X scale, the x-distance corresponding
                         to the width of a pixel.  Must be
                         non-zero.

Null separator: 1 byte

Yunit          yu bytes  Latin-1 symbol or description of the
                         height unit, eg. meters, degrees North
                         Latitude,  etc).  A zero-length string
                         may be used if the data is
                         dimensionless.

Null separator: 1 byte

Yoffset        yo bytes  Y offset, a real number written as a
                         text floating-point value.  This is
                         the physical value of "y"
                         corresponding to the top edge of the
                         image.
```

```
Null separator: 1 byte

Yscale         ys bytes  Y scale, the y-distance corresponding
                         to the height of a pixel.  Must be
                         non-zero.
```

There is no trailing zero for the final string.

The scaling algorithm for finding the physical x-value and y-value of the middle of a pixel is

```
physical_x_value := Xoffset + Xscale * (column_number + 0.5)
physical_y_value := Yoffset + Yscale * (row_number + 0.5)
```

If present, the `xySC` chunk must appear before the first `IDAT` chunk. Only one instance of the `xySC` chunk is permitted in a PNG stream.

4.8 `zsCL` Z-Scale of Pixel Values

When a PNG file is being used to store physical data other than color values, such as a two-dimensional temperature field, the `zsCL` chunk can be used to record the relationship between pixel values and actual physical values. The `zsCL` data might be used to construct a reference color bar beside the image. It is not expected to affect the way the image is displayed.

The `zsCL` chunk's contents are

```
Equation type   1 byte   (0 for linear scaling,
                          1 for base-e exponential scaling,
                          2 for arbitrary-base exponential
                            scaling)

N               1 byte   (Number of parameters)

Unit            u bytes  (Latin-1 symbol or description of
                          the unit, eg. K, Population Density,
                          MPa, etc).  A zero-length string may
                          be used if the data is dimensionless.

Null separator: 1 byte

P0             p0 bytes  (First parameter, a real number
                          written as a text floating-point
                          value [link to Floating-Point Values
                          in extensions document]

Null separator: 1 byte
```

```
P1              p1 bytes  (Second parameter)

Null separator: 1 byte    (only if P2 appears)

P2              p2 bytes  (Third parameter, optional)
```

There is no trailing zero for the final string.

The scaling algorithm is

```
if equation_type == 0 then
scaled_value := P0 + P1 * normalized_sample_value

else if equation_type == 1 then
scaled_value := P0 + P1 * EXP(P2 * normalized_sample_value)

else if equation_type == 2 then
scaled_value := P0 + P1 * P2^normalized_sample_value
```

in which EXP(x) is the exponential function, e^x, in which "e" is the base of natural logarithms (approximately 2.718282). For color_types 0 and 4, the normalized sample value is the grayscale sample value of the pixel normalized to the range [0.0:1.0] by dividing it by the maximum value for the bit depth, using floating point arithmetic.

For color_types 2, 3, and 6, the scaling algorithm is applied independently to each of the color sample values similarly normalized to [0.0:1.0].

Pure logarithmic data can be expressed either with

```
Equation_type = 1                      Equation_type = 2
N  = 3                    or with      N  = 3
P0 = 0                                 P0 = 0
P1 = min                               P1 = min
P2 = LOGe(max/min)                     P2 = max/min
```

The `zsCL` data is not intended to be used by a decoder to affect the way the pixels are displayed. `zsCL` is simply a comment, and could be used, for example, to construct a reference color bar scale beside the image, or to extract the original physical values recorded in the file.. The `drNG/DrNG` or `loGE/LoGE` chunk should be used if the encoder wants the decoder to modify the sample values for display purposes.

If present, the `zsCL` chunk must appear before the first `IDAT` chunk. Only one instance of the `zsCL` chunk is permitted in a PNG stream.

5 Security considerations

Security considerations are addressed in the basic PNG specification, `http://www.w3.org/pub/WWW/TR/WD-png`.

The same precautions taken when displaying `tEXt` data should be taken when displaying the text contained in the "unit" strings of the `xySC` and `zsCL` chunks and in the "name" string of the `spLT` chunk. Viewers should not display these strings directly without first checking for the presence of nonprintable characters, and for the <ESC> character in particular.

No known additional security hazards are posed by the chunks described here.

6 Credits

Contributors

Contributors' names are presented in alphabetical order:

- Mark Adler, madler@alumni.caltech.edu
- Thomas Boutell, boutell@boutell.com
- Christian Brunschen, cb@df.lth.se
- Adam M. Costello, amc@cs.wustl.edu
- Lee Daniel Crocker, lee@piclab.com
- Andreas Dilger, adilger@enel.ucalgary.ca
- Oliver Fromme, fromme@rz.tu-clausthal.de
- Todd French, todd@lfd.physics.uiuc.edu
- Jean-loup Gailly, gzip@prep.ai.mit.edu
- Chris Herborth, chrish@qnx.com
- Alex Jakulin, alex@hermes.si
- Neal Kettler, kettler@cs.colostate.edu
- Jim King, jimk@mathtype.com
- Tom Lane, tgl@sss.pgh.pa.us
- Alexander Lehmann, alex@hal.rhein-main.de
- Chris Lilley, chris.lilley@mcc.ac.uk
- Dave Martindale, davem@cs.ubc.ca

- Owen Mortensen, ojm@csi.compuserve.com
- Robert P. Poole, lionboy@primenet.com
- Glenn Randers-Pehrson, glennrp@arl.mil or randeg@alumni.rpi.edu
- Greg Roelofs, newt@uchicago.edu
- Willem van Schaik, gwillem@ntuvax.ntu.ac.sg
- Guy Schalnat, schalnat@group42.com
- Paul Schmidt, pschmidt@photodex.com
- Tim Wegner, 71320.675@compuserve.com

Editor

- Glenn Randers-Pehrson, glennrp@arl.mil or randeg@alumni.rpi.edu

End of PNG Proposed Chunks Listing