Portable Network Graphics Frame Format Proposal Version 0.960727

File

draft-pnf-0.960727.txt

Status of this Memo

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

It is a **proposal**, and the format is subject to change.

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/mpng/`.

Notices

Changes from seventh draft (draft-pnf-960720)

- Moved background definition from CHDR to FHDR chunk and combined it with the "transparency is present" field. Removed definition of "subimage 0" as the background in composite frames. These changes will allow the viewer to start up with a clean slate at each seek point ("initial" frame) instead of with litter left over from previous frames in the stream.

- Made it legal to omit the FHDR chunk and accept all default values for FHDR fields, provided that the first chunk is IHDR or PHDR.
- Removed some of the "[]"-delimited questions (should we allow block deltas?; other delta_types?)

Abstract

This document presents the *[proposed]* the format of a PNF (Portable Network graphics Frame) stream. PNF is a multiple-image format that can contain simple animations (slide shows) comprised of PNG single-image streams. It will also incorporate PNP (Portable Network Photo) images, once a definition for PNP is developed, and incorporates delta and composite frames, which are defined herein.

It also presents a *[proposed]* definition for the format of a PND (Portable Network graphics Delta) stream, which defines an image in terms of an original PNG, PNP, or PND image and the differences from that image.

It also presents the *[proposed]* format of a single-image, composite frame stream, whose subimages are PNG, PND, or PNP single-image streams. A composite frame can be independent, or, if it contains PND streams, it can depend upon the information in the previous frame.

The PNF format uses the same chunk structure that is defined in the PNG specification, and shares other features of the PNG format.

Contents

1 Introduction

This *[proposed]* specification defines the format of the PNF (Portable Network graphics Frame) stream. It also defines the format of the PND (Portable Network graphics Delta) stream and the format of a single-image, composite frame stream. PND and the composite image stream can be components of a PNF stream.

Note: This *[proposed]* specification depends on the PNG Portable Network Graphics) and PNP (Portable Network Photograph) specifications. PNP is under discussion by `pnp-list@dworkin.wustl.edu`. The PNG specification is available at the PNG home page,

    http://quest.jpl.nasa.gov/PNG/

A PNF stream describes a sequence of single images, each of which can be composed of one or more PNG, PNP, PND (PNG-Delta, defined herein), or composite images.

PNF is pronounced "Ping-frame."

When the PNF stream is a standalone file, use the PNG 8-byte header with "PNF" instead of "PNG" in bytes 2-4, and ".pnf" as the file suffix.

PNF does not accommodate sound or complex sequencing information, nor does it accommodate playing a stream backwards. Those are left to be described in the MPNG specification, which is yet to be defined, and is being discussed in the `mpng-list@dworkin.wustl.edu` mailing list.

Chunk structure (length, name, CRC) and the chunk-naming system are identical to those defined in the PNG specification. As in PNG, all integers that require more than one byte will be in network byte order.

A PNF stream consists of a `FHDR` chunk, followed by one or more image definitions, followed by the `FEND` chunk. The first image can be a PNG stream (`IHDR`, PNG chunks, `IEND`), a PNP stream `PHDR`, PNP chunks, `PEND`), or an "initial" composite frame (`CHDR`, composite frame chunks, `CEND`). Subsequent images can be any of these, or a PND stream (`PHDR`, PND chunks, `PEND`), or a "dependent" composite frame. Each chunk of the PNF stream or of any image definition is an independent entity, i.e., no chunk is enclosed in the data segment of another chunk.

2 Top-level PNF chunks

2.1 Critical PNF chunks

2.1.1 `FHDR` PNF stream header

```
4 bytes: max_frame_width  (unsigned integer,
         default 2^31-1)

4 bytes: max_frame_height (unsigned integer,
         default 2^31-1)

4 bytes: max_number_of_frames (unsigned integer,
         default 2^31-1)  There are not more than
         max_number_of_frames in this PNF stream.

4 bytes: max_chunk_length (unsigned integer,
         default 2^31-1)  No chunk in this stream,
         including in any included PNGs or PNPs, has
         a data field exceeding this length.

4 bytes: ticks_per_second  (unsigned integer, default 1)
```

```
4 bytes: frame_duration (unsigned integer, default 1) in
         ticks.  The time between the beginning of
         displaying one frame until the beginning of
         displaying the next.

1 byte:  max_bit_depth (unsigned integer, default 16)

1 byte:  background_source (unsigned integer, default 0)
         0: use the supplied background color
         1: read the current display contents or application
            specified background if possible, but if not
            possible, then use supplied background color
         2: background is not required.

2 bytes: red_background (unsigned integer, default 0)

2 bytes: green_background (unsigned integer, default 0)

2 bytes: blue_background (unsigned integer, default 0)

1 byte:  ok_to_discard (unsigned integer, default 0)
           0: image data must be retained
           1: image data may be discarded after
              processing each frame.  This is a promise
              that this PNF stream contains no PNDs or
              dependent composite frames)
```

Any of the FHDR data fields can be set equal to zero, meaning that the value assumes the default value. Encoders are strongly encouraged not to use zeros, especially for the max_frame_width and max_frame_height, though, if the information is available when the encoder is writing the FHDR chunk.

The FHDR chunk may be omitted, provided that the first chunk in the stream is IHDR or PHDR. In this event, decoders must assume the default values for all of the FHDR data.

2.1.2 FEND End of PNF stream

The FEND chunk's data length is zero. It signifies the end of a PNF stream.

2.1.3 IHDR, PNG chunks, IEND

A PNG (Portable Network Graphics) stream.

See the PNG specification for the format of the PNG chunks.

Any chunks between `IHDR` and `IEND` are written and decoded according to the PNG specification. The image width and height must not exceed `max_frame_width` and `max_frame_height` from the `FHDR` chunk.

2.1.4 `DHDR`, PND chunks, `DEND`

A PND (PNG-Delta) stream.

See Chapter 3, The PND Format, below, for the format of the PND stream. Any chunks between `DHDR` and `DEND` are written and decoded according to the PND specification, below. The image width and height must not exceed `max_frame_width` and `max_frame_height` from the `FHDR` chunk.

2.1.5 `PHDR`, PNP chunks, `PEND`

A PNP (Portable Network Photo) stream.

See the PNP specification for the format of the `PHDR` chunk.

[It is assumed that the PNP stream will begin with a PHDR chunk and end with a PEND chunk, and it is assumed that the first two fields of PHDR will be unsigned 4-byte integers giving the image width and height. It is also assumed that the PNP image data will be contained in a sequence of PDAT chunks.]

The image width and height must not exceed `max_frame_width` and `max_frame_height` from the `FHDR` chunk.

Any chunks between `PHDR` and `PEND` are written and decoded according to the PNP specification. Until PNP is defined, it is acceptable for viewers to display an empty width by height frame when a PNP stream is encountered. Such viewers should, when encountering a `PHDR` chunk, simply skip chunks until a `PEND` chunk is found.

2.1.6 `CHDR` composite frame chunks, `CEND`

A composite frame stream. See Chapter 4, The composite frame, below, for the format of the composite frame stream.

2.1.7 `SEEK` Seek point

A seek point.

```
n bytes: previous (number of bytes since the
         previous SEEK chunk) If previous == 0,
         then the number of bytes is unspecified.
```

```
n bytes: next (number of bytes to the next SEEK
         chunk) If next == 0, then the number of
         bytes is unspecified.

n is the length of the SEEK chunk, divided by two.
         n must be either 4 or 8.
```

The SEEK chunk is only allowed at positions in the PNF stream where a restart is possible, and no information appearing prior to the SEEK chunk (other than the information in the FHDR chunk) is required to display the remainder of the stream properly. In addition to providing a mechanism for skipping frames or backspacing over frames, this provides a means of dealing with a corrupted datastream. The viewer would abandon processing and simply look for the next SEEK chunk before resuming. Note that looking for a PNG IHDR or PNP PHDR would not be sufficient because the PNG or PNP stream might be a subimage a of a composite frame rather than a complete frame description.

When n is eight, 32-bit machines will have to interpret "previous" as a set of two integers, the first representing the number of complete 4G blocks and the second (the last four bytes of "previous") as the remainder, and will have to treat "next" similarly.

"Previous" and "next" are measured from the first length byte of one SEEK chunk to the first length byte of another SEEK chunk.

Applications are allowed to forget everything preceding the SEEK chunk, except for data appearing in the FHDR chunk, so new instances of any needed chunks are required. The first frame following the SEEK chunk must not be defined by a PND stream or a "dependent" composite frame.

Multiple instances of the SEEK chunk are permitted.

2.2 Ancillary PNF chunks

2.2.1 dISC Ok to discard

The dISC chunk temporarily overrides the "ok_to_discard" field in the FHDR chunk, when "ok_to_discard" is zero. The dISC chunk's data length is zero. The presence of the chunk signifies that decoders may discard the subsequent PNG, PND, PNP, or composite frames after processing, and can rest assured that no PND or dependent composite frames will follow. The effect of the dISC chunk remains until a SEEK chunk is encountered. When the dISC chunk is not in effect, then decoders must keep the image data for possible use in a subsequent PND or dependent composite frame, if the "ok_to_discard" FHDR field is zero. Note that it is not possible to override "ok_to_discard" when "ok_to_discard" is not zero.

Multiple instances of the dISC chunk are permitted.

2.2.2 dURa Duration

Duration of display.

```
4 bytes: duration (unsigned integer), in ticks, using
         the tick length determined from ticks_per_second
         defined in the FHDR chunk.
```

Overrides the value of duration given in the FHDR chunk. The value of "duration" will remain in effect until another "dURa" chunk is encountered or until a "SEEK" chunk is encountered, when the duration reverts to the value from FHDR. Multiple instances of the dURa chunk are permitted, but no more than one dURa chunk is permitted between any two critical chunks.

2.2.3 tEXt, zTXt Text chunks

Same as PNG.

3 The PND format

A PND stream describes a single image, by giving the changes from a previous PNG (Portable Network Graphics), PNP (Portable Network Photo), or another PND image. While it is possible for a PND stream to be stored in a standalone file (where the original image is available in another standalone file), a PND stream will normally be found as a component of a PNF stream or a MPNG (Multiple Portable Network Graphics, yet to be defined) stream.

If the PND stream is a standalone file (not usually the case), use the PNG 8-byte header with "PND" instead of "PNG" in bytes 2-4, and use ".pnd" as the file suffix.

PND is pronounced "Ping-delta."

The decoder must have available an original (decoded) image from which the original chunk data is known. How the original image is made available to the decoder is up to the implementation. It could be in a separate file or it could be provided earlier in a PNF datastream. The original image can be the result of decoding a PNG, a PNP, or another PND stream. If alpha or transparency is present, then the decoder must also have available the original background against which the original image was displayed, if transparency or alpha are to be composited against the background.

The decoder must not have modified the pixel data in the original image by applying output transformations such as gAMA or cHRM, or by compositing the image against a background. Instead, the decoder must make available to the PND decoder the unmodified pixel data along with the values for the gAMA, cHRM, and any other recognized chunks from the original image datastream.

A PND stream consists of a DHDR and DEND enclosing other optional chunks (if there are no other chunks, the decoder simply copies the original image).

Chunk structure (length, name, CRC) and the chunk-naming system are identical to those defined in the PNG specification. Definitions of compression_type, filter_type, and interlace_type are also the same as defined in the PNG specification.

3.1 PND critical chunks

3.1.1 DHDR PND stream header

```
4 bytes: width
   Must match original image if IDAT or PDAT is not present.
   When IDAT or PDAT is present, the original pixel data is
   discarded entirely and replaced by a new image with
   dimensions width x height.

4 bytes: height
  Must match original image if IDAT or PDAT is not present

1 byte: bit_depth
  Must match bit_depth of original image if IDAT or PDAT is
  not present.

1 byte: color_type
  Must match original image if IDAT or PDAT is not present.

       0, 2, 3, 4, 6: same as PNG
       8: PNP grayscale image
      10: PNP color image
      12: PNP grayscale image with alpha
      14: PNP color image with alpha
```

[or whatever PNP developers decide]

```
1 byte: compression_type
  Same as PNG
  Need not match original image.

1 byte: filter_type
  Same as PNG
  Need not match original image.

1 byte: interlace_type
  Same as PNG
  Need not match original image.

1 byte: delta_type
  0: subtraction, by bytes, modulo 256.
```

```
1 byte: copy-safe
  0: This stream contains no changes that would render
     unknown chunks unsafe-to-copy.  All chunks will be
     copied from the previous image unless they are
     mentioned in a DELE chunk.
  1: This stream contains changes that would render
     unknown chunks unsafe-to-copy.  Such chunks will not
     be copied from the previous image unless they are
     mentioned in a COPY chunk.
```

3.1.2 `DEND` End of PND stream

End of PND stream. This chunk has a zero data length.

3.1.3 `IDAT` PNG image header

New image data in PNG format. Just as in PNG, multiple, consecutive IDAT chunks are allowed, whose contents, when concatenated, form a zlib datastream containing PNG pixel data. `IDAT` is not allowed if `PDAT` or `DDAT` is present.

3.1.4 `PDAT` PNP image data

New image data in PNP format, yet to be defined.

Until the PNP `PDAT` chunk is defined, it is acceptable for a PND decoder to skip this chunk.

Multiple, consecutive `PDAT` chunks are allowed, whose contents, when concatenated, form a PNP image description. PDAT is not allowed if `IDAT` or `DDAT` is present.

3.1.5 `DDAT` Pixel deltas

Deltas from an original image known to the decoder. `DDAT` is not allowed if `IDAT` or `PDAT` is present.

If `DDAT` is present, then the image width and height, color_type, and bit_depth must match those of the original image, and the original image must have been derived from a PNG stream or from a sequence of PND streams that depend upon a PNG stream, but not from a PNP stream.

Similarly to the PNG `IDAT` chunk, there can be multiple, consecutive `DDAT` chunks whose contents are concatenated to make up a zlib datastream that, when decompressed, contains a filtered and perhaps interlaced set of delta pixel samples. The actual pixel values are calculated using the method defined in delta_type field of the `FHDR` chunk.

When delta_type is 0, an encoder calculates the bytes in the DDAT chunk by subtracting the new image bytes from the original image bytes, modulo 256. When decoding the DDAT chunk, the new image bytes are obtained by adding the delta bytes to the original image bytes, modulo 256. This is similar in operation to the PNG SUB filter.

When color_type is 3, the deltas are differences between index values (by bytes), not between color samples.

3.1.6 `COPY` Chunks to be copied

Chunks to be copied

```
chunkname1
...
chunknamen
```

`COPY` provides a list of chunknames that are to be copied (values inherited) from the original image, regardless of the copy-safe rules, and regardless of whether the chunk is recognized or not. The number of names is determined from the chunk length, divided by 4.

There can be multiple instances of the `COPY` chunk. A `COPY` chunk appearing before `IDAT` can be used to copy any incoming chunks. A `COPY` chunk appearing after `IDAT` can only be used to copy chunks appearing after `IDAT` in the original image.

The `COPY` chunk does not specify the placement of chunks within the resulting PNG stream; it merely marks them for copying. The decoder can place them in any order, relative to their order in the original image, that obeys the chunk-ordering rules for copy-safe chunks given in the PNG specification.

Note that the `COPY` chunk data itself is inherited (unless "COPY" appears in a `DELE` chunk), so a `COPY` chunk need only appear in the first of a sequence of PND streams, if there are no changes.

If a chunkname appears in a `COPY` chunk and also appears in a `DELE` chunk, the last instance takes precedence. If the last instance appears after the `IDAT`, `PDAT`, or `DDAT` chunk, it will only affect chunks appearing after `IDAT`, `PDAT`, or `DDAT` chunk.

It is not an error for a chunkname to appear in the copy list, when that chunk does not appear in the original image.

3.1.7 `DELE` Chunks not to be copied

Chunks to be deleted

```
chunkname1
...
chunknamen
```

DELE provides a list of chunknames that are not to be copied (values not to be inherited) from the original image, regardless of the copy-safe rules, and regardless of whether the chunk is recognized or not. The number of names is determined from the chunk length, divided by 4.

There can be multiple instances of the DELE chunk. When a DELE chunk appears before IDAT, all instances of the named chunks will be marked for deletion. A DELE chunk appearing after IDAT can only be used to delete chunks appearing after IDAT in the original image.

If a chunkname appears in a COPY chunk and also appears in a DELE chunk, the last instance takes precedence. If the last instance appears after the IDAT, PDAT, or DDAT chunk, it will only affect chunks appearing after IDAT, PDAT, or DDAT chunk.

It is not an error for a chunkname to appear in the DELE list, when that chunk does not appear in the original image.

Note that the DELE chunk data itself is inherited (unless "DELE" appears in a DELE chunk), so a DELE chunk need only appear in the first of a sequence of PND streams, if there are no changes.

3.2 Other PND chunks

Any PNG or PNP chunk can appear in a PND stream, and are handled as described in the governing specification. When only one instance of the chunk is permitted, the chunk data replaces the chunk data from the original file. When multiple instances are permitted, the chunk is treated as an additional instance. When multiple instances are not permitted, the chunk data replaces the original data. When the format depends on the color_type, the appropriate format must be used.

Note that a gAMA, cHRM, or similar chunk existing in the original image would not affect the pixel data inherited by this PND stream because they are not used in decoding the pixel data. Applications are responsible for ensuring that the pixel values that are inherited from the previous image have not been transformed in any way after decompressing and unfiltering them.

When processing the oFFs and pHYS chunks, the viewer is responsible for providing the correct background if alpha or transparency is present; when the image size or location changes, the background will not be the same background against which the original image was composited.

When processing the tRNS chunk, if color_type is 3 and PLTE is not supplied, then the number of allowable entries is determined from the number of PLTE entries in the original image.

4 The composite frame

A composite frame is a single image, made up of a background and zero or more subimages, each defined by a PNG, PNP, or PND stream. The LOCA chunk can be used to specify the placement of each subimage within the composite frame.

A composite frame can be an "initial" frame or a "dependent" frame. If it is "dependent" then it requires information inherited from the previous frame, such as the background and any PNG, PNP, or PND subimages that are to be redisplayed or that are to serve as the basis for a PND subimage. PNF viewers that process composite frames must be prepared to "remember" any number of such subimages from the immediately previous composite frame.

PNF viewers are expected to recognize and process the PNG `pHYs` and `oFFs` chunks encountered in subimages, even though these are ancillary chunks. When the `pHYs` chunk appears with unit_specifier=0, then that image should be scaled to to obtain the desired aspect ratio by scaling the image height and leaving the image width fixed.

4.1 Critical composite frame chunks

4.1.1 `CHDR` Composite frame header

Beginning of a composite frame.

```
4 bytes: composite_frame_width  (unsigned integer)
         (must match that of the previous frame if
         frame_type = 1, and must not exceed
         max_frame_width in the FHDR chunk)

4 bytes: composite_frame_height  (unsigned integer)
         (must match that of the previous frame if
         frame_type = 1, and must not exceed
         max_frame_height in the FHDR chunk)

4 bytes: x_dimension (unsigned integer), default 0,
         corresponding to composite_frame_width, in
         micrometers.  If non-zero, this value is to be
         used when processing pHYs and oFFs chunks
         found in subimages, even if the actual frame
         dimensions are otherwise available.

4 bytes: y_dimension (unsigned integer), default 0,
         corresponding to composite_frame_height.


1 byte:  frame_type
         0: Initial frame
         1: Dependent frame
```

4.1.2 `CEND` End of composite frame

End of composite frame. This chunk has a zero data length.

4.1.3 LOCA Subimage location

Location for the following subimage.

The LOCA chunk gives the position, measured downward and to the right of the upper left corner of the frame, where the following subimage is to be located.

The chunk's contents are:

```
4 bytes: Image position, X axis (signed integer)
4 bytes: Image position, Y axis (signed integer)
```

Note that negative values are permitted, and denote displacement in the opposite directions. Note that LOCA can specify an image placement which is partially or wholly outside the frame boundaries. In such cases, the resulting subimage must be clipped to fit within the frame, or not displayed at all if it falls entirely outside the frame.

If the subimage contains an oFFs chunk, the subimage's offset is computed with respect to the position defined by the LOCA chunk (convert the oFFs distances to pixel units, and add them to the image position defined by LOCA).

After processing one subimage, the offset values revert to (0,0) until another LOCA chunk is encountered.

4.1.4 IHDR, PNG chunks, IEND

The image width and height are allowed to exceed those given in the CHDR chunk; if so, the image must be clipped to fit them.

4.1.5 DHDR, PND chunks, DEND

The image width and height are allowed to exceed those given in the CHDR chunk; if so, the image must be clipped to fit them.

4.1.6 PHDR, PNP chunks, PEND

The image width and height are allowed to exceed those given in the CHDR chunk; if so, the image must be clipped to fit them.

4.1.7 SODE Subimage source and destination

```
1 byte: source
        0: none (subimage will be completely defined
           in this frame by a PNG or PNP stream)
        1: subimage defined in previous frame
        2: subimage previously defined in this frame
```

```
1 byte: destination
        0: discard immediately after displaying subimage
        1: keep (make available for use by next PNF)
        2: discard after completion of this PNF

2 bytes: source ID (unsigned integer)
        when source=0: omitted
        when source=1: subimage identified by
           "destination ID" in the previous frame
        when source=2: subimage identified by
           "destination ID" earlier in this frame

2 bytes: destination ID (unsigned integer)
        when destination=0: omitted
        when destination=1: the "source ID" by which
            this subimage can be identified later in
            this frame or in the following frame
        when destination=2: the "source ID" by which
            this subimage can be identified later in
            this frame.
        Can be the same as the destination ID of a
            previous subimage, if the previous subimage
            will no longer be required.
```

If a `SODE` chunk is not followed by a subimage description (`IHDR`, `DHDR`, or `PHDR`) (i.e. a `FEND` chunk or another `SODE` chunk is encountered prior to encountering any `IHDR`, `DHDR`, or `PHDR` chunk), then the subimage from the previous frame with the specified source ID is given the specified destination ID for possible use in the following frame, but not displayed in the present frame.

If a subimage header chunk (`IHDR` or `PHDR`) is not preceded by a `SODE` chunk (i.e. no `SODE` chunk was encountered after the preceding `FHDR`, `IEND`, `PEND`, or `DEND` chunk), the chunk will be treated as if it had been preceded by a `SODE` chunk with source=0, destination=0, source ID and destination ID omitted.

The `DHDR` chunk introducing a PND stream must be preceded by a `SODE` chunk with a valid source ID.

4.2 Ancillary composite frame chunks

4.2.1 gPLT Global Palette

Same as proposed PNG `spLT`; a global suggested palette inherited by all subimages. Multiple `gPLT` chunks are allowed within a composite frame description. If present, they must appear prior to any `IHDR`, `DHDR`, or `PHDR` chunk in the composite frame.

4.2.2 oFFs Composite frame offset

Same as PNG, offsets the entire composite frame. If a subimage also contains an oFFs chunk, the values from both oFFs chunks are added together with the LOCA value to form a total offset for the subimage.

Only one oFFs chunk is allowed within a composite frame description, and, if present, it must appear prior to any IHDR, DHDR, or PHDR chunk in the composite frame.

4.2.3 tEXt, zTXt Text chunks

Same as PNG.

5 Security Considerations

Security considerations are addressed in the basic PNG specification. No known additional security concerns are raised by this format.

Detection of corrupted file transfers can be improved beyond that available in PNG by using the FHDR max_chunk_size field to determine whether any chunk length (except for that of FHDR itself, which has a known length that can be checked) is unreasonably large.

6 Appendix: Examples

6.1 Example 1: simple movie

Here's an example simple movie PNF comprised of PNGs and PNDs:

```
\211 P N F \r \n ^z \n  # PNF signature
FHDR 720 468  # width and height
       0 0    # no of frames, max chunk length unspecified
       30  3  # 10 frames per second
       8   1  # 8-bits, no alpha or transparency
       0      # Not OK to discard
tEXtTitle\0Sample Movie

SEEK
IHDR 720 468 ...
gAMA 77000    # It's a good practice to include gAMA always
IDAT ...
IEND
```

```
DHDR 720 468 ...# a PNG-delta frame
DDAT ...
DEND

DHDR 720 468 ...# another PNG-delta frame
DDAT ...
DEND

SEEK            # ok to restart here
IHDR 720 468    # complete PNG frame follows
gAMA 77000      # The gamma from the first frame has
IDAT ...        #    been forgotten so need new one
IEND

DHDR 720 468 ...# another PNG-delta frame
DDAT ...
DEND

FEND            # end of PNF stream
```

6.2 Example 2: single composite frame

Here's an example single-composite-frame PNF, which takes a grayscale image and draws it side-by-side with a false-color version of the same image:

```
\211 P N F \r \n ^z \n # PNF signature
FHDR 1024 512 1 8192 # width, height, nframes, maxchunklen
     0 0      # tick length, duration unspecified
     16 1 1   # depth 16, no transparency, ok to discard
CHDR 1024 512 0 0    # Initial composite frame, 1024 x 512
     50 50 150 2 1 0 # Define a blue background, don't try
                     #  to read and save existing background
LOCA 6 6             # Location of first subimage
SODE 0 2 1           # New object 1, keep for later use
IHDR 500 500 16 0 .. # A graylevel image (note, no PNG
gAMA 50000           # signature here)
IDAT ...
IEND                 # End of subimage

SODE 2 0 1           # Reload object 1, discard afterwards
LOCA 518 6           # Location for second subimage.  Note
                     #    that SODE-LOCA order doesn't matter.
DHDR 500 500 16 0 .. # Png-delta, inherits gAMA and IDAT
                     #    from object 1
tEXtComment\0The faLS chunk is described in ftp://swrinde....
faLS ...             # Apply pseudocolor to previous subimage
                     # No DDAT or IDAT; pixels don't change
DEND                 # End of subimage
```

```
LOCA 900 400         # Overlay near lower right-hand corner
IHDR 101 101 2 3 ... # SODE can be omitted when the subimage
                     #    doesn't depend on another and
                     #    nothing depends on it.
gAMA 50000           # We need a new gAMA, though, because
PLTE ...             #    there's no "source ID" for object.
tRNS ...             # It's transparent (maybe a logo)
IDAT ...             # Note that the color type can differ
IDAT ...             #    from that of the other objects.
IEND                 # End of subimage

CEND                 # End of composite frame
FEND                 # End of PNF stream
```

6.3 Example 3: movie with sprites

Here's another movie, illustrating the use of PND streams as sprites

```
\211 P N F \r \n ^z \n  # PNF signature
FHDR 512 512 ... # Start of PNF stream
CHDR ...         # First frame
SEEK             # establish SEEK point
LOCA x1 y1       # Location for object 1
SODE 0 1 1       # define object 1  and export to next frame
PHDR ...         # it's a PNP image
gAMA 50000
PDAT ...
PEND
LOCA x2 y2       # Location for object 2
SODE 0 1 2       # define object 2 and export to next frame
IHDR ...         # It's a PNG
gAMA 50000
IDAT ...
IEND
CEND             # end of frame
```

```
CHDR ...        # Next frame (repeat this CHDR-CEND
                #   sequence with different locations to
                #   move the objects around)
LOCA x1 y1      # new location for object 1
SODE 1 1 1 1    # retrieve object 1 from previous frame,
                #    export to next
DHDR            # A completely empty DHDR-DEND just
                #    redisplays the object
DEND            # at the new location
LOCA x2 y2      # new location for object 2
SODE 1 1 2 2    # retrieve object 2 from previous frame,
                #    export it to the next
DHDR            # another empty DHDR-DEND
DEND
LOCA x3 y3      # Location for another copy of object 2
SODE 2 1 2 3    # reuse object 2 from this frame,
                #    export it as object 3
DHDR            # another DHDR-DEND
tRNS ...        # make it semitransparent
DEND
CEND            # end of frame

FEND            # end of PNF stream
```

7 Credits

Contributors' names are presented in alphabetical order:

- Mark Adler, madler@alumni.caltech.edu
- Thomas Boutell, boutell@boutell.com
- John Bowler, jbowler@megamed.com
- Christian Brunschen, cb@df.lth.se
- Glen Chapman, glenc@clark.net
- Adam M. Costello, amc@cs.wustl.edu
- Lee Daniel Crocker, lee@piclab.com
- Peter da Silva, peter@starbase.neosoft.com
- Andreas Dilger, adilger@enel.ucalgary.ca
- Oliver Fromme, fromme@rz.tu-clausthal.de
- Jean-loup Gailly, gzip@prep.ai.mit.edu

- Chris Herborth, chrish@qnx.com
- Alex Jakulin, alex@hermes.si
- Neal Kettler, kettler@cs.colostate.edu
- 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
- 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
- Paul Schmidt, pschmidt@photodex.com
- Smarasderagd, smar@reptiles.org
- Thomas R. Tanner, ttehtann@argonet.co.uk
- Guido Vollbeding, guivol@esc.de
- Tim Wegner, 71320.675@compuserve.com
- Alaric B. Williams, png@abwillms.demon.co.uk

Editor

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

End of PNF Specification Expires 27 January 1997