PNG-GROUP-DRAFT
MNG
Expires: 07 Mar 1997

PNG Development Group
mpng-list@dworkin.wustl.edu
07 Sep 1996

MNG (Multiple Network Graphics) Format Version 0.960907

File

draft-mng-960907.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/mng/documents/`.

Changes from fourteenth MNG draft (draft-mng-960902)

- Changed pixel delta calculations to be by samples instead of by bytes.
- Revised the semantics of the COPY and NOCO chunks. COPY also serves as a template for a generated PNG.
- Added SAVE chunk and modified SEEK chunk description.
- Separated the FRAM chunk into FRAM and CLIP chunks.
- Combined the BLOC and DHDR chunks.
- Added DEFN chunk.
- Reinstated the BACK chunk and moved the relevent fields out of the MHDR chunk.
- Added max_samples_per_pixel field to MHDR.

- Rewrote first paragraph of Section 4, Retaining Image Data.
- Made it an error to set the LOCA loca_delta_type == 1 when there is no basis image.
- Added loop_effect field to Examples 4 and 6.
- pSIZ refers to max_frame_width and max_frame_height, not frame_width and frame_height
- Eliminated the term "subimage" and just used "image" throughout.
- Changed "stream" to "datastream" throughout.
- Rewrote the LSHO chunk description.
- Added NEED and NONE chunk descriptions.
- Included Adam Costello's EDIT proposal.

Abstract

This document presents the *[proposed]* the format of a MNG (Multiple Network Graphics) datastream. MNG is a multiple-image extension of the PNG (Portable Network Graphics) format, that can contain animations (slide shows) comprised of PNG single-image datastreams. It can also incorporate images in a highly compressible "PND" format, defined herein.

The MNG format provides a mechanism for reusing image data without having to retransmit it. Multiple images can be composed into a "frame," and an image can be used as a "sprite" that moves from one location to another in subsequent frames.

A MNG frame normally contains a two-dimensional image or a two-dimensional layout of smaller images. It could also contain three-dimensional "voxel" data arranged as a series of two-dimensional planes (or tomographic slices), each plane being represented by a PNG or PND datastream.

A PND datastream defines an image in terms of a basis PNG or PND image and the differences from that image. This has been demonstrated to provide a much more compact way of representing subsequent images than using a complete PNG datastream for each.

The MNG format uses the same chunk structure that is defined in the PNG specification, and share other features of the PNG format. Any valid PNG datastream is also a valid MNG datastream.

This document includes a number of examples that demonstrate various capabilities of MNG including simple movies, composite frames, loops, fades, tiling, scrolling, and storage of voxel data.

Contents

1 Introduction 3
2 MNG chunks 3
2.1 Critical MNG chunks 4

2.1.1 MHDR MNG datastream header . . . 4
2.1.2 MEND End of MNG datastream . . . 6
2.1.3 LOCA Image location . . . 6
2.1.4 BACK Background . . . 6
2.1.5 CLIP Image clipping boundaries . . . 7
2.1.6 IHDR, PNG chunks, IEND . . . 8
2.1.7 DHDR, PND chunks, DEND . . . 8
2.1.8 DEFI, DEFN Define an image for reuse . . . 8
2.1.9 FRAM and ENDF Compose a frame . . . 9
2.1.10 CLON Clone an image . . . 9
2.1.11 SHOW Show an image . . . 9
2.1.12 DISC Discard images . . . 10
2.1.13 SAVE Save decoder state . . . 10
2.1.14 SEEK Seek point . . . 11
2.1.15 LOOP, ENDL Define a loop . . . 11
2.1.16 LSHO Loop SHOW . . . 12
2.1.17 SYNC Synchronize display . . . 13
2.1.18 NONE Define a blank image . . . 13
2.1.19 NEED Resources neede . . . 14
2.2 Ancillary MNG chunks . . . 14
2.2.1 dURA Duration . . . 14
2.2.2 gPLT Global Palette . . . 14
2.2.3 pSIZ Physical Image Size . . . 16
2.2.4 tEXt, zTXt, tIME Text, Time chunks . . . 16
3 The PND format . . . 16
3.1 PND critical chunks . . . 17
3.1.1 DHDR PND datastream header . . . 17
3.1.2 DEND End of PND datastream . . . 18
3.1.3 PROM Promotion of basis image . . . 18
3.1.4 IHDR, PNG chunks, IEND . . . 19
Image replacement . . . 19
Image pixel deltas . . . 20
Image alpha deltas . . . 20
No change to pixel data . . . 20
3.1.5 COPY Chunks to be copied . . . 20
3.1.6 NOCO Chunks not to be copied . . . 21
3.2 PND ancillary chunks . . . 22
3.2.1 fADE Fade in or out . . . 22
3.2.2 tEXt, zTXt, tIME Text, Time chunks . . . 22
4 An alternate proposal for PND . . . 23
5 Retaining image data . . . 27
6 Decoder handling of fatal errors . . . 27
7 Decoder handling of interlaced files . . . 27

8 Decoder handling of palettes 27
9 Security Considerations 28
10 Appendix: Examples 28
10.1 Example 1: Simple movie 28
10.2 Example 2: Single composite frame 29
10.3 Example 3: Movie with sprites 30
10.4 Example 4: “Fading in” a transparent image 31
10.5 Example 5: Storing three-dimensional images 32
10.6 Example 6: Tiling 33
10.7 Example 7: Scrolling 34
11 Credits 35

1 Introduction

This *[proposed]* specification defines the format of the MNG (Multiple Network Graphics) datastream.

Note: This *[proposed]* specification depends on the PNG Portable Network Graphics specification. The PNG specification is available at the PNG home page,

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

A MNG datastream describes a sequence of single frames, each of which can be composed of one or more images definded by PNG or PND (PNG-Delta, defined herein) datastreams.

MNG is pronounced “Ming.”

The first eight bytes of a MNG datastream are

```
138 77 78 71 13 10 26 10
```

which is similar to the PNG signature with “\212 M N G” instead of “\211 P N G” in bytes 1-4. Use “.mng” as the file suffix.

MNG does not yet accommodate sound or complex sequencing information, nor does it accommodate playing a datastream backwards. These capabilities might be added at a later date, in a backwards-compatible manner. These issues are being discussed in the `mpng-list@dworkin.wustl.edu` mailing list. At some future date, support for the PNP (Portable Network Photo) format might be added. PNP is under discussion by `pnp-list@dworkin.wustl.edu`.

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 MNG datastream consists of the MNG signature, and a `MHDR` chunk, followed by one or more frame definitions, followed by the `MEND` chunk. The first frame must be a PNG datastream (`IHDR`, PNG chunks, `IEND`) or a group of image definitions (including at least one PNG datastream) enclosed in a `FRAM, ENDF` pair.

Each subsequent frame can be a PNG datastream, a PND datastream (DHDR, PND chunks, DEND), a SHOW chunk, or a group of image definitions and SHOW chunks enclosed in a FRAM, ENDF pair. Each chunk of the MNG datastream or of any image definition is an independent entity, i.e., no chunk is ever enclosed in the data segment of another chunk.

An independent PNG datastream, with a PNG signature, is also a valid MNG datastream that must be recognized and decoded by MNG-compliant decoders. This kind of MNG datastream must contain only a single image.

2 MNG chunks

This section describes chunks that can appear at the top level of a MNG datastream. Unless otherwise specified in the PND section of this specification, they need not be recognized there.

2.1 Critical MNG chunks

2.1.1 MHDR MNG datastream header

The MHDR chunk is always first in all MNG datastreams eccept for those that consist of a PNG datastream with a PNG signature. The MHDR chunk contains 49 bytes:

```
4 bytes: max_frame_width  (unsigned nonzero integer)
         Maximum width of any image or frame to be
         displayed

4 bytes: max_frame_height (unsigned nonzero integer)
         Maximum width of any image or frame to be
         displayed

4 bytes: max_stored_image_width  (unsigned nonzero
         integer) Maximum width of any image that must
         be stored

4 bytes: max_stored_image_height (unsigned nonzero
         integer) Maximum height of any image that must
         be stored

4 bytes: max_number_of_frames (unsigned integer)
         There are not more than max_number_of_frames
         in this MNG datastream.  If this field is
         zero, max_number_of_frames is undefined.
```

```
4 bytes: max_chunk_length (unsigned nonzero integer)
         No chunk in this datastream, including in
         any included PNGs, has a data field
         exceeding this length.  If this field is zero,
         max_chunk_length is undefined.

4 bytes: ticks_per_second  (unsigned nonzero integer)

4 bytes: frame_duration (unsigned integer) in
         ticks.  The desired minimum amount of time
         to elapse between the beginning of displaying
         one frame until the beginning of displaying
         the next.

4 bytes: total_duration (unsigned integer) in ticks.
         Maximum total duration of the entire
         datastream.
         The sum of the individual frame_durations
         (including all instances of frames that are
         displayed as a consequence of processing the
         LOOP chunk) must not exceed this value (under
         actual playback conditions the display is likely
         to take longer).  If this field is zero, the
         maximum total duration is undefined.

4 bytes: default_gamma (unsigned integer)
         The value of gamma, times 100000, to be assumed
         for any images in the datastream that do not
         supply their own value of gamma.  This default
         gamma value also applies to the background color
         or application-supplied background image, if the
         gamma value for the background is unknown, and
         is the gamma value to be assumed when using the
         gPLT chunk.  If this field is zero, the
         default_gamma is undefined.

1 byte:  max_bit_depth (unsigned nonzero integer)
         No image in this MNG datastream has (or is
         promoted to an image having) a greater bit
         depth.  Legal values are 1, 2, 4, 8, or 16.

1 byte:  max_samples_per_pixel (unsigned nonzero integer)
         No image in this MNG datastream has (or is
         promoted to an image having) more than this
         number of samples per pixel.  Legal values are
         1, 2, 3, or 4 (PNG images with color_type == 3
         have one sample per pixel).
```

```
1 byte:  ok_to_discard (unsigned integer)
         0: image data of each image must be retained
            until it is explicitly discarded with the
            DISC chunk.
         1: image data can be discarded after processing
            each image.  This is a promise that this MNG
            datastream contains no PND datastreams or
            SHOW or LSHO chunks.

6 bytes: Reserved, must be zero
         [or some other reasonably small number of bytes]
```

2.1.2 MEND End of MNG datastream

The MEND chunk's data length is zero. It signifies the end of a MNG datastream.

2.1.3 LOCA Image location

Location for the following image.

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

The chunk's contents are:

```
1 byte:  loca_delta_type (unsigned integer)
         0: LOCA data gives X and Y directly
         1: LOCA positions are determined by adding
            the LOCA data to the position of the
            basis image
4 bytes: Image position, X axis (signed integer)
4 bytes: Image position, Y axis (signed integer)
```

Negative values are permitted, and denote displacement in the opposite directions. LOCA can specify an image placement that is partially or wholly outside the display boundaries. In such cases, the resulting image must be clipped to fit within the display, or not displayed at all if it falls entirely outside the display. The display boundaries are taken from the max_frame_width and max_frame_height fields of the MHDR chunk (or from the boundaries given in the CLIP chunk, if present).

If the image contains an oFFs chunk, the image'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).

[Remove this paragraph about oFFs? Say something else instead?]

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

If there is no basis image, it is an error to set `loca_delta_type == 1`. If there is a basis image but no `LOCA` chunk, the new image is displayed at the same location as the basis image.

2.1.4 `BACK` Background

The `BACK` chunk defines the background against which transparent, clipped, or less-than-full-frame images can be displayed.

```
1 byte:  background_source (unsigned integer)
         0: Viewers can use the supplied background color
            as a default.
         1: Viewers must use the supplied background
            color.
         2: Viewers must use the image with image_id == 0
            as the background for all subsequent images.
            When background_source == 3, this image must
            not be transparent, its width and height
            must be max_frame_width and max_frame_height
            from the MHDR chunk, and the ok_to_discard
            field of the MHDR chunk must be zero.

2 bytes: red_background (unsigned integer)

2 bytes: green_background (unsigned integer)

2 bytes: blue_background (unsigned integer)
```

Viewers are expected to composite every frame in the MNG datastream, whether it be a PNG or PND datastream or a group of PNG or PND datastreams enclosed in a (`FRAM`, `ENDF`) pair, against a fresh copy of the background.

Multiple instances of the `BACK` chunk are permitted in a MNG datastream. One of these should appear before the `SAVE` chunk, if the `SAVE` chunk is present.

The `BACK` chunk can be omitted. If a background is required and the `BACK` chunk is omitted, then the viewer must supply its own background.

2.1.5 `CLIP` Image clipping boundaries

This chunk gives the boundaries to which images must be clipped for display. It contains four 4-byte fields:

```
4 bytes: left_clip (unsigned integer)
4 bytes: right_clip (unsigned integer).  Must be
         greater than left_clip and less than or
         equal to max_frame_width from the IHDR
         chunk.
4 bytes: top_clip (unsigned integer)
4 bytes: bottom_clip (unsigned integer).  Must be
         greater than top_clip and less than or
         equal to max_frame_height from the IHDR
         chunk.
```

The clipping boundaries remain in effect until another CLIP chunk or a SEEK chunk is encountered. If the CLIP chunk appears prior to the SAVE chunk, then it also gives the clipping boundaries that are to be restored upon encountering a SEEK chunk. When no CLIP chunk is in effect, the clipping boundaries are taken from the MHDR chunk:

```
left_clip   := 0
right_clip  := max_frame_width
top_clip    := 0
bottom_clip := max_frame_height
```

2.1.6 IHDR, PNG chunks, IEND

A PNG (Portable Network Graphics) datastream.

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_image_width and max_image_height from the MHDR chunk. After applying the LOCA offset, the image must be clipped to fit the max_frame_width> and max_frame_height> limits from the MHDR (or from the boundaries given in the CLIP chunk, if present).

2.1.7 DHDR, PND chunks, DEND

A PND (PNG-Delta) datastream.

See Chapter 3, The PND Format, below, for the format of the PND datastream. Any chunks between DHDR and DEND are written and decoded according to the PND format. The image width and height must not exceed max_image_width and max_image_height from the MHDR chunk. After applying the LOCA offset, the image must be clipped to fit the max_frame_width> and max_frame_height> limits from the MHDR (or from the boundaries given in the CLIP chunk, if present).

It is an error for the DHDR chunk to appear when the ok_to_discard field in the MHDR chunk is nonzero.

2.1.8 DEFI, DEFN Define an image for reuse

The DEFI chunk defines an image and immediately displays it. The DEFN chunk defines an image without displaying it.

```
2 bytes: image_id (unsigned integer) image identifier
         to be given to the image that immediately
         follows the DEFI or DEFN chunk.  Subsequent
         DHDR, SHOW, CLON, LSHO, and DISC chunks can
         use this number to identify it.
```

If image_id is an identifier that already exists, the basis image previously associated with the identifier is discarded.

If a IHDR–IEND sequence is not immediately preceded by a DEFI or DEFNchunk, then "DEFI 0" is implied, unless the "ok_to_discard" field of the MHDR chunk is set.

2.1.9 FRAM and ENDF Compose a frame

Beginning of a composite frame. A composite frame is made up of one or more images defined by PNG or PND datastreams or SHOW or LSHO chunks.

The FRAM chunk is empty.

The LOCA chunk can be used to specify the placement of each image within the frame. If the images are transparent or do not cover the entire frame, as defined by the max_frame_width and max_frame_height fields of the MHDR chunk, they are composited against the background defined by the BACK chunk, or against an application-defined background, if the BACK chunk is not present.

Viewers are expected to ignore the frame_duration value while inside the FRAM, ENDF pair and display all of the images at once, if possible, or as fast as can be managed. The frame_duration value is the desired minimum time to elapse from the beginning of displaying the first image of the frame until the beginning of the next image after the entire frame.

When images in a frame overlap, viewers are expected to composite the later images against the partially completed frame that includes all earlier images.

The ENDF chunk marks the end of a composite frame. This chunk is empty. For every FRAM chunk, there must be a corresponding ENDF chunk later in the MNG datastream, and for every ENDF chunk, there must be a corresponding FRAM chunk earlier in the MNG datastream.

2.1.10 CLON Clone an image

Create a clone (a new copy) of an image. The clone is initially identical to the original image and has the same location. Subsequent chunks can modify the clone without affecting the original image, or they can modify the original without affecting the clone.

```
2 bytes: image_id (unsigned integer) identifier of the
         image to be cloned.

2 bytes: clone_id (unsigned integer) identifier to be
         to be given to the clone (new copy) of the
         image.  Subsequent DHDR, SHOW, and DISC
         chunks can use this number to locate it.
```

2.1.11 SHOW Show an image

Retrieve and display a previously-defined image, without making any changes.

```
2 bytes: source_id (unsigned integer) image identifier
         by which a previously-defined image is to be
         retrieved and displayed.
```

An instance of the image will be displayed at the location specified by the LOCA chunk. This is exactly equivalent to a DHDR, DEND sequence that makes no changes to the image.

It is not necessary to follow an IHDR-IEND or DHDR-DEND sequence with a SHOW chunk. Such images are always displayed if they are within clipping boundaries of the frame, unless the sequence was preceded by the DEFN chunk.

2.1.12 DISC Discard images

This chunk can be used to inform the decoder that it can discard the image data associated with the associated image identifiers.

The chunk contains a list of image identifiers. If the chunk is empty, all images except the image whose image_id == 0 can be discarded. Image 0 can be discarded by explicity including 0 in the list.

The chunk contains a sequence of zero or more two-byte identifiers. The number of images to be discarded is the the chunk's data length, divided by two.

```
2 bytes: discard_id (unsigned integer) image identifier
         that can be discarded.  All information
         pertaining to the corresponding image can be
         disarded and the identifier can be reused by
         a DEFI chunk.
etc.
```

The appearance of an image_id in the discard_id list, when no such image has been stored, should not be treated as a fatal error.

2.1.13 SAVE Save decoder state

The SAVE chunk is empty.

It appears after the set of chunks that define the decoder state that must be restored upon encountering a SEEK chunk.

Only one image can be defined ahead of the SAVE chunk. If such an image does appear ahead of the SAVE chunk, it must have image_id == 0, and it is not permitted to redefine, modify, relocate, or discard the image with image_id == 0 later in the MNG datastream.

Only one instance of the SAVE chunk is permitted in a MNG datastream.

2.1.14 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 MNG datastream where a restart is possible, and no information appearing prior to the SEEK chunk (other than the information in the MHDR chunk and information appearing ahead of the SAVE chunk, if present) is required to display the remainder of the datastream 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 chunk would not be sufficient because the PNG datastream might be inside a loop or might need data from preceding LOCA or CLIP chunks.

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 MHDR chunk and anything appearing ahead of the SAVE chunk, if the SAVE chunk is present.

The SEEK chunk is not permitted within the scope of a LOOP, ENDL pair. If a decoder encounters a SEEK chunk while any loop is active, either as a result of an illegal SEEK chunk appearing inside a loop or as the result of skipping corrupted data, all display loops are immediately terminated.

Multiple instances of the SEEK chunk are permitted.

2.1.15 LOOP, ENDL Define a loop

The LOOP chunk provides a "shorthand" notation that can be used to avoid having to repeat identical chunks in a MNG datastream. Its contents are

```
1 byte: start_loop_level (unsigned integer)
1 byte: loop_effect (unsigned integer)
        0: Execution of the loop might modify or relocate
           images.
        1: Execution of the loop might modify or relocate
           images, but upon completion of the loop, all
           images have been restored to their initial
           state and location.
        2: Execution of the loop does not modify or
           relocate any images.
4 bytes: repeat_count (unsigned integer) range 0 to
           2^31-1
```

Decoders must treat the chunks enclosed in a loop exactly as if they had been repeatedly spelled out. Therefore, during the first iteration of the loop, the basis images for any PND datastreams in the loop are the images in existence prior to entering the LOOP chunk, but in subsequent interations these basis images might have been modified. The loop_effect field can be used to inform decoders that it is safe to reduce the number of loop iterations or to replay the images in the loop without recompositing them.

When the LOOP chunk is present, an ENDL chunk with the same loop_level must be present later in the MNG datastream. Loops can be nested. Each inner loop must have a higher value of than the loop that encloses it.

If repeat_count is zero, the loop is done zero times. Upon encountering a LOOP chunk with repeat_count == 0, decoders simply skip chunks until the matching ENDL chunk is found, and resume processing with the chunk immediately following it.

It is the responsibility of the encoder to make sure that the assertions made by the loop_effect field are true.

The ENDL chunk ends a loop that begins with the LOOP chunk. It contains a single one-byte field:

```
1 byte: end_loop_level (unsigned integer) range 0 to 255
```

When the ENDL chunk is encountered, the loop repeat_count is decremented. If the result is nonzero, processing resumes at the beginning of the loop. Otherwise processing resumes with the chunk immediately following the ENDL chunk.

When the ENDL chunk is present, a LOOP chunk with the same loop_level must be present earlier in the MNG datastream.

2.1.16 LSHO Loop SHOW

The LSHO chunk can be used to create a loop that shows a series of images. It contains two two-byte fields

```
2 bytes: first_id (unsigned integer)
2 bytes: last_id  (unsigned integer)
4 bytes: repeat_count (unsigned integer) range
            0 to 2^31-1
```

If first_id > last_id then the frames are shown in reverse order.

```
LSHO first_id last_id n
```

is exactly equivalent to

```
LOOP 255 2 n
SHOW first_id
SHOW next_id
etc
SHOW last_id
ENDL 255
```

2.1.17 SYNC Synchronize display

The SYNC chunk is empty.

The SYNC chunk provides a point at which the processor must wait for all pending displays to finish up before resuming, perhaps because of a need to synchronize a sound datastream (not defined in this specification) with the display, to synchronize stereo images, and the like.

2.1.18 NONE Define a blank image

[Maybe BLNK would be a better chunk name] The NONE chunk defines a blank image. The chunk data is the same as that of the PNG IHDR chunk:

```
4 bytes: width (unsigned integer)
4 bytes: height (unsigned integer)
1 byte:  bit_depth (unsigned integer)
1 byte:  color_type (unsigned integer)
1 byte:  compression_method (unsigned integer)
1 byte:  filter_type (unsigned integer)
1 byte:  interlace_type (unsigned integer)
```

The NONE chunk defines a PNG image with a set of IHDR variables, that can be used as a basis image by subsequent PND datastreams.

It generates a rectangle with zeroes in all of the pixel samples, which represents a black rectangle, fully transparent if the color type is 4 or 6. If color_type is 3, it also generates a PLTE of the appropriate length, filled with zeroes.

2.1.19 NEED Resources neede

The NEED chunk can be used to specify needed resources, to provide a quick exit path for viewers that are not capabable of displaying the MNG datastream.

The NEED chunk contains a list of chunk names that the decoder must be prepared to encounter.

```
4 bytes: chunkname_1
etc.
4 bytes: chunkname_n
```

The number of chunk names is determined from the chunk length, divided by 4.

The NEED chunk should be placed early in the MNG datastream, preferably immediately after the MHDR chunk. Viewers not recognizing critical chunk names in the list should abandon the MNG datastream or, if the unrecognized chunk name is ancillary, can display a warning or request user intervention.

2.2 Ancillary MNG chunks

2.2.1 dURA Duration

Duration of display, the minimum time that must elapse from the beginning of displaying one frame until the beginning of displaying the next.

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

Overrides the value of duration given in the MHDR 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 MHDR or SAVE. Multiple instances of the dURA chunk are permitted, but no more than one dURA chunk is permitted between any two critical chunks. The dURA chunk takes effect with the beginning of the next image to be displayed. The dURA chunk is not permitted inside a FRAM, ENDF pair.

2.2.2 gPLT Global Palette

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

The format of this chunk is identical to that of the [proposed] sPLT PNG chunk. The 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. "256 color including Macintosh default", "256 color including Windows-3.1 default", "50-color rgb palette for use with early versions of Mosaic") identifies the palette, which can permit applications or people to choose the appropriate one when more than one suggested palette appears in a MNG datastream. The "name" string must consist only of printable ASCII characters and must not have leading or trailing blanks, but can have single embedded blanks. There must be at least one and no more than 79 characters in the name. Names are case-sensitive. Decoders should filter out any nonprintable characters, especially the ESC character, in the "name" string before displaying it, to avoid possible security hazards.

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.

Each frequency entry is proportional to the approximate fraction of pixels in the images 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, meaning the color is "least important" or that it is rarely if ever used.

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, as described under "bit depth rescaling" in the PNG specification.

If the file gamma value for an image is different from the default gamma value from the MHDR chunk, decoders will need to gamma-correct the image samples before quantizing them to the gPLT palette.

Multiple gPLT chunks, with different names, are allowed in a MNG datastream. If present, they must appear prior to any IHDR or DHDR chunk to which they apply. The gPLT chunk can appear for any color type. When an image contains a PLTE or sPLT suggested palette, the gPLT data takes precedence. The gPLT chunks should be placed ahead of the SAVE chunk, if the SAVE chunk is present, to ensure that the gPLT data is not lost when a SEEK chunk is encountered.

2.2.3 pSIZ Physical Image Size

This chunk provides values for the physical image size that viewers can use when processing pHYS and oFFs chunks found in images, when the viewer does not have a better idea.

```
4 bytes: x_dimension (unsigned integer),
         corresponding to max_frame_width, in
         micrometers.

4 bytes: y_dimension (unsigned integer),
         corresponding to max_frame_height.
```

Only one instance of the pSIZ chunk is permitted. If it appears, it must appear prior to the PNG or PND datastreams to which it pertains, and should be placed prior to the SAVE chunk, if the SAVE chunk is present.

MNG viewers should (but are not required to) recognize and process the PNG pHYs and oFFs chunks encountered in images, 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.

2.2.4 tEXt, zTXt, tIME Text, Time chunks

The tEXt, zTXt, and tIME chunks are the same as those in PNG.

3 The PND format

A PND datastream describes a single image, by giving the changes from a previous PNG (Portable Network Graphics) or another PND image. The PND format might be extended at some later date to include a PNP (Portable Network Photo) datastream.

No provision is made for storing a PND datastream as a standalone file. A PND datastream will normally be found as a component of a MNG datastream. Applications that need to store a PND datastream separately can wrap it in a MNG datastream consisting of the MNG signature, the MHDR chunk, a NONE chunk, the PND datastream, and a MEND chunk.

The decoder must have available a basis (decoded) image from which the original chunk data is known. The basis image can be the result of decoding a PNG or another PND datastream.

The new image is always of the same basic type (at present only PNG is defined) as the basis image.

The decoder must not have modified the pixel data in the basis 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 basis image datastream.

A PND datastream consists of a DHDR and DEND enclosing other optional chunks (if there are no other chunks, the decoder simply copies the basis 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 datastream header

```
2 bytes: image_id (unsigned integer)
         Identifies the basis image from which changes
         will be made, and is also the image_id
         of the resulting basis image for a subsequent
         PND datastream.

1 byte: image_type
         0: Image type is unspecified.  An IHDR chunk
            must be present.
         1: Image type is PNG.   IHDR can be omitted
            if no IHDR fields are different from those
            in the basis image and delta_type is 0, 1,
            or 3, and if IHDR would otherwise have
            appeared immediately after DHDR.
```

```
1 byte: delta_type
         0: image replacement
         1: pixel addition, by samples, modulo
            2^bit_depth.
         2: alpha addition, by samples, modulo
            2^bit_depth.  Regardless of the color
            type of the basis image, the IDAT data
            are written as a grayscale image (color
            type 0) but the decoded samples are used
            as deltas to the alpha samples in the
            basis image.  The basis image must have
            (or be promoted to via the PROM chunk)
            color type 4 or color type 6.
         3: no change to pixel data

4 bytes: block_width (unsigned integer)

4 bytes: block_height (unsigned integer)

4 bytes: block_x_location (unsigned integer) measured
            in pixels from the left edge of the basis
            image.

4 bytes: block_y_location (unsigned integer) measured
            in pixels from the top edge of the basis
            image.

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

The image type, whether given explicitly as 1 or 2 or implied by the presence of an `IHDR` chunk, must be the same as that of the basis image.

The `block_width` and `block_height` fields give the size of the block of pixels to be modified or replaced, and `block_x_location` and `block_y_location` give its location with respect to the top left corner of the basis image. The block must fall entirely within the basis image.

The block size and location fields are ignored `delta_type` is 3.

3.1.2 DEND End of PND datastream

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

3.1.3 PROM Promotion of basis image

This chunk is used to "promote" a basis image to a higher bit depth or to add an alpha channel, before making changes to it.

```
1 byte: new color_type
1 byte: new bit_depth
```

The PROM chunk must appear ahead of the IHDR chunk, if IHDR is present, and ahead of any chunks that would have followed IHDR, if IHDR is omitted.

When a grayscale or truecolor basis image is promoted to an image with an alpha channel, the alpha samples are initialized to 2^bit_depth-1 (fully opaque). When an indexed-color image is promoted, the alpha channel data is obtained from the basis image's tRNS chunk data, if present, or initialized to 2^bit_depth-1, if the tRNS chunk is not present or not recognized by the decoder.

The PROM chunk is not permitted to "demote" a basis image to an image with a lesser bit depth or from one with an alpha channel to one without an alpha channel.

The resulting image must not exceed max_bit_depth or max_samples_per_pixel from the MHDR chunk.

3.1.4 IHDR, PNG chunks, IEND

A partial PNG (Portable Network Graphics) datastream. The basis image must be a PNG or PNG-based PND.

See the PNG specification for the format of the PNG chunks. The PNG datastream must contain at least IHDR and IEND but can inherit other chunk data from the basis image. Except for IDAT, any chunks appearing between IHDR and IEND are always treated as replacements or additions and not as deltas.

A gAMA, cHRM, or similar chunk existing in the basis image would not affect the pixel data inherited by this PND datastream because they are not used in decoding the pixel data. Applications are responsible for ensuring that the pixel values that are inherited from the basis 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 basis 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 basis image.

The IHDR chunk can be omitted if all of the IHDR fields would be identical to those in the basis image, the image_type is 1, and the IHDR chunk would otherwise have appeared immediately after the DHDR chunk with no intevening PND chunks such as COPY. The decoder must treat this datastream as though the IHDR were present, immediately after the DHDR chunk, with all IHDR chunk data identical to that of the basis image. When delta_type is 2, it is not necessary to include an IHDR chunk for the sole purpose of specifying that the IDAT is in grayscale format.

When the IHDR chunk is present and delta_type is nonzero, its width and height must match match those of the basis image. Also, when the IHDR chunk is present, the block height and width must match the height and width of the basis image, and the block location must be (0,0).

The PNG specification places ordering requirements on many chunks with respect to the PLTE and IDAT chunks. If neither of these two chunks is present, and the COPY chunk is not present, all unknown chunks appearing in the partial PNG datastream are considered to have appeared before both. Known chunks (including all standard chunks described in the PNG specification) are considered to have appeared in their proper order with respect to the critical chunks. When the COPY chunk is present, then all chunks are considered to have appeared in the order given by the COPY chunk.

The IEND chunk can be omitted, if it would be the last chunk in the PND datastream before the DEND.

Image replacement When the delta_type == 0 in the DHDR chunk, the pixel data in the IDAT chunks represent a completely new image. In ths case, none of the IHDR data need match that of the basis image.

Image pixel deltas When the delta_type == 1 in the DHDR chunk, the pixel data in the IDAT chunks represent deltas from the pixel data in a basis image known to the decoder.

The image color_type, and bit_depth must match those of the basis image, and the basis image must have been derived from a PNG datastream or from a sequence of PND datastreams that depend upon a PNG datastream.

The compression method, filter method, and interlace method need not be the same.

[We might want to add a compression_type that uses deflate with the basis image data as a preset dictionary]

The IDAT chunk data contains a filtered and perhaps interlaced set of delta pixel samples. The delta samples are presented in the order specified by interlace method, filtered according to the filter method and compressed according to the compression method given in the IHDR chunk. The actual pixel values are calculated using the method defined in the delta_type field of the DHDR chunk.

When delta_type == 1, an encoder calculates the new sample values from the samples in the basis image and those in the new image by subtracting the basis image samples from the new image samples, modulo 2^sample_bit_depth. When decoding the IDAT chunk, the new image bytes are obtained by adding the delta bytes to the basis image bytes, modulo 2^sample_bit_depth. This is similar in operation to the PNG SUB filter, except that it works by samples instead of by bytes.

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

Image alpha deltas When the `delta_type == 2` in the `DHDR` chunk, the pixel data in the `IDAT` chunks represent deltas from the alpha data in a basis image known to the decoder. The color samples are not changed, and the updated alpha samples are calculated in the same manner as the updated pixel samples are calculated when `delta_type == 1`.

The image bit_depth must match those of the basis image, but the color type must be 0 (grayscale). The basis image must have an alpha channel or must have been promoted to a type that has an alpha channel. The compression method, filter method, and interlace method need not be the same.

No change to pixel data When the `delta_type == 3` in the `DHDR` chunk, there is no change to the pixel data. If `IHDR` is present, the color_type in the `IHDR` data must match that of the basis image.

When the `delta_type == 3` in the `DHDR` chunk, It is an error for `IDAT` to appear.

3.1.5 `COPY` Chunks to be copied

Lists chunks to be copied and provides a template for an output PNG datastream.

```
4 bytes: chunkname_1
etc.
4 bytes: chunkname_n
```

The `COPY` chunk provides a list of chunknames that are to be copied values inherited) from the basis 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.

The `COPY` chunk also suggests the placement of chunks within the resulting PNG datastream. The decoder can place them in any order, relative to their order in the basis image, that obeys the chunk-ordering rules for copy-safe chunks given in the PNG specification, but when it encounters an unknown but copy-safe chunk, it should use the `COPY` chunk data as a template to ensure that it is properly placed.

There can be only one instance of the `COPY` chunk in a PND datastream. The `COPY` chunk must appear before `IHDR` if it is present.

The `COPY` chunk data itself is inherited (unless "COPY" appears in a `NOCO` chunk), so a `COPY` chunk need only appear in the first of a sequence of PND datastreams, if there are no changes. When a `COPY` chunk is inherited and also appears on the PND datastream, the entire list of chunk names is replaced.

If a chunkname appears in the `COPY` chunk and also appears in a `NOCO` chunk, the `NOCO` chunk takes precedence, and the appearance in the `COPY` chunk only establishes the placement of any new copy of the chunk that might appear in the PND datastream.

Applications that reconstruct a PNG datastream from a PND datastream and a basis PNG or PND datastream should not write the `COPY` chunk to the resulting PNG datastream, because the `COPY` chunk would not be recognized by a PNG or MNG decoder.

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

3.1.6 NOCO Chunks not to be copied

Chunks not to be copied

```
4 bytes: chunkname_1
etc.
4 bytes: chunkname_n
```

The NOCO chunk provides a list of chunknames that are not to be copied (values not to be inherited) from the basis 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.

If a chunkname appears in a COPY chunk and also appears in a NOCO chunk, the NOCO chunk takes precedence, and its appearance in the COPY chunk only establishes the placement of any new copy of the chunk that might appear in the PND datastream.

There can only be one instance of the NOCO chunk in a PND datastream. The NOCO chunk must appear before IHDR if it is present.

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

Applications that reconstruct a PNG datastream from a PND datastream and a basis PNG or PND datastream should not write the NOCO chunk to the resulting PNG datastream.

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

3.2 PND ancillary chunks

3.2.1 fADE Fade in or out

This chunk can be used to "fade" an image in or out against the background, without having to transmit new alpha values.

```
1 byte: (unsigned integer) fade_type
      0: fade out
      1: fade in
      2: fade in but don't change fully transparent
         pixels
2 bytes: (unsigned integer) alpha_delta
```

When "fade_type" is 0, the value of alpha_delta is subtracted from the alpha sample of every pixel in the basis image, but the result is not allowed to fall below zero.

When "fade_type" is 1, the value of alpha_delta is added to the alpha sample of every pixel in the basis image, but the result is not allowed to exceed the maximum alpha value for the image's bit depth.

When "fade_type" is 2, the value of alpha_delta is added to the alpha sample of any pixel in the basis image that has a non-zero value, and the result is not allowed to exceed the maximum alpha value for the image's bit depth.

If color_type is 3, then the value of alpha_delta is added to or subtracted from the alpha values that were defined by the tRNS chunk data in the basis image, and the resulting values become the tRNS data exported to any subsequent image. The maximum alpha value for this color_type is 255, regardless of the bit depth.

If color_type is 0 or 2, the fADE chunk is ignored.

If IHDR is also present, the fADE chunk must appear before IHDR. The fade operation is performed on the image data after decoding the chunks between IHDR and IEND.

3.2.2 tEXt, zTXt, tIME Text, Time chunks

The tEXt, zTXt, and tIME chunks are the same as those in PNG.

4 An alternate proposal for PND

This PND proposal was submitted by Adam "Seven" Costello.

[Note that for testing purposes, a PND with a 22-byte header is written according to this alternate format, while a 21-byte header is written as above. Of course only one proposal will survive.]

```
DHDR
====

subimage ID (2 bytes):
```

[Should this be moved up out of PND, into MNG somewhere?]

```
delta type (1 byte):
    0   pixel replacement
    1   pixel addition
    2   alpha addition
    3   no change
```

[Note that I changed the addition/subtraction terminology, not the actual meaning, because I think it's more intuitive this way. I'm telling the decoder what it must do with the delta values and the old pixel values: add them.]

```
compression method (1 byte)
interlace method   (1 byte)
filter method      (1 byte)
    As in IHDR.

x_offset (4 bytes)
y_offset (4 bytes)
width    (4 bytes)
height   (4 bytes):
    The area to be modified.  Ignored if
    delta_type is 3.

DEND
====

Empty.

DDAT
====
```

Like IDAT. The interpretation of DDAT is affected by all the fields in DHDR (except subimage ID). The bit depth and color type are not given in DHDR because they are required to agree with the ones in IHDR (after any changes are made to IHDR by the PND). DDAT must appear if and only if delta type is not 3.

```
EDIT
====

Contains zero or more 5-byte records with the format:
```

```
command (1 byte):
     0        move to just before first occurrence
     1        move to just after first occurrence
     2        move to just before last occurrence
     3        move to just after last occurrence
     4        move to just before next occurrence
     5        move to just after next occurrence
     6        move to just before previous occurrence
     7        move to just after previous occurrence
     8 or  9  delete first occurrence (and move there)
    10 or 11  delete last occurrence (and move there)
    12 or 13  delete next occurrence (and move there)
    14 or 15  delete previous occurrence (and move there)
    24 or 25  delete all occurrences (and move to last)
    26 or 27  delete all occurrences (and move to first)
    28 or 29  delete all subsequent occurrences (and
              move to last)
    30 or 31  delete all prior occurrences (and move
              to first)

name (4 bytes):
    chunk name to search for

Note that the command byte can be interpreted as
bit-flags:

    bit 0:  just before vs. just after
    bit 1:  search forward vs. search backward
    bit 2:  search from start/end of file vs. search
            from current position
    bit 3:  move vs. move & delete
    bit 4:  perform once vs. repeat until failure
            (ignored if bit 3 is unset)
```

[I can't help noticing that bit 4 is ignored when bit 3 is unset, and bit 0 can be ignored when bit 3 is set, so bits 0 and 4 could be overlaid. I find the idea appealing and disgusting at the same time.] The result would be:

```
 0..7     unchanged
 8        delete first occurrence (and move there)
 9        delete all occurrences (and move to last)
10        delete last occurrence (and move there)
11        delete all occurrences (and move to first)
12        delete next occurrence (and move there)
13        delete all subsequent occurrences (and move
          to last)
14        delete previous occurrence (and move there)
15        delete all prior occurrences (and move to
              first)

Alternate format for the command field:

command (2 bytes):
    byte 1:   A: Add after
              I: Insert before
              D: Delete once
              R: Remove repeatedly
    byte 2:   F: First
              L: Last
              N: Next
              P: Previous

IF        move to just before first occurrence
AF        move to just after first occurrence
IL        move to just before last occurrence
AL        move to just after last occurrence
IN        move to just before next occurrence
AN        move to just after next occurrence
IP        move to just before previous occurrence
AP        move to just after previous occurrence
DF        delete first occurrence (and move there)
DL        delete last occurrence (and move there)
DN        delete next occurrence (and move there)
DP        delete previous occurrence (and move there)
RF        delete all occurrences (and move to last)
RL        delete all occurrences (and move to first)
RN        delete all subsequent occurrences (and move
          to last)
RP        delete all prior occurrences (and move to
          first)
```

If RF and RL look backwards, remember that RF means “remove first, repeatedly”, so you end up at the position of the last.

The chunk name in an EDIT record must not be IDAT. If a PND changes IHDR, it must not change the width and height. The PND decoder is responsible for coercing the IDAT data to conform to the other IHDR fields.

```
KEEP
====
```

May optionally appear once before any EDIT or PNG chunks (i.e. right after DHDR). It contains zero or more chunk names. All chunks in the original PNG which do *not* match any of the names are deleted, and will not be seen by EDIT chunks. The remaining chunks can still be deleted by EDIT chunks.

```
SAFE
====
```

May occur anywhere between DHDR and DDAT, and may occur more than once. It contains a list of unsafe-to-copy chunk names which should be considered safe-to-copy by the PND decoder.

```
PND
===
```

In regular expression syntax,

PND is DHDR KEEP? (EDIT — SAFE — <PNG-chunk>)* DDAT* DEND

A PND decoder processes a PND and a basis PNG to produce a resultant PNG as follows:

Copy the basis PNG to the work area, which will ultimately become the resultant PNG. Initialize the current position pointer to the start of the work area. Process chunks from the PND in order. Chunks other than EDIT/KEEP/SAFE are simply inserted into the work area at the current position. Records of EDIT chunks are processed as implied above, in order of appearance. Note that searches do *not* wrap around from the end to the start (nor from the start to the end). If the chunk sought is not found, the record is a no-op. Therefore, PND authors should sometimes precede a search that might fail with a move to a known location.

Once a DDAT or DEND chunk is reached, we know we are past all EDIT chunks, so we know the final state of IHDR. Do any coercion necessary on the IDAT data. Then DDATs, if they appear, can be processed and applied to the image.

After DEND is encountered, if any critical changes have been made, fix all unsafe-to-copy chunks surviving from the original PNG (remember to take into account any SAFE chunks that were seen). For unknown chunks, “fix” means remove. For known chunks, “fix” can mean alter and/or relocate, or remove.

A PND decoder cannot operate on a basis PNG that contains an unknown critical chunk.

Note that this is a conceptual model, and a real PND decoder is likely to have digested the basis PNG into an internal format and forgotten the original chunks. Nevertheless, it can reconstruct an order–the order that it *would* output chunks *if* it were going to output a PNG based on its internal representation. It can interpret the EDIT commands in the context of this reconstructed order. Note that it doesn’t have to reconstruct the chunks themselves–just their order. And it can pretend that it would output only one big IDAT.

But no matter what sort of representation the PND decoder has for the basis PNG, it must have random access to it. It only needs sequential access to the PND stream.

5 Retaining image data

When ok_to_discard == 0 in the MHDR chunk, the decoder must retain information about each image for possible redisplay with the SHOW or LSHO chunks or for possible use as the basis image for a subsequent PND datastream. The LOCA data associated with an image must also be retained, in addition to its pixel data and the values associated with other recognized PNG chunks such as PLTE and gAMA, subject to the chunk copying rules and the COPY and NOCO chunks and the copy-safe field of the DHDR chunk.

When the encoder knows that image data will not be needed by subsequent frames, it can make life easier for decoders by using the ok_to_discard field of the MHDR chunk or by using the DISC chunk.

When the encoder knows that all subsequent images will completely fill the display and do not use transparency, the background_source field of the MHDR chunk can be used to inform the decoder that it does not have to retain the background.

6 Decoder handling of fatal errors

When a fatal error is encountered, such as an unknown critical chunk in an image or a missing basis image where one was required, MNG viewers should attempt to recover gracefully by abandoning processing of the frame and searching for a SEEK chunk.

MNG editors, on the other hand, should be more strict and reject any file with errors unless the user intervenes.

7 Decoder handling of interlaced files

Decoders are required to be able to interpret datastreams that are interlaced, but are only required to display the completed frames; they are not required to display the images as they evolve. Viewers that are decoding datastreams coming in over a slow communication link might want to do that, but MNG authors should not assume that the frames will be displayed in other than their final form.

8 Decoder handling of palettes

When a PLTE chunk is received, it does not affect the display of any previous image in the datastream.

If PLTE is present in a PND datastream, the new palette is used in displaying the image defined by the PND; if no IDAT chunk is present and the image type is PNG indexed-color, then the image is redisplayed using the old pixel samples as indices into the new palette.

If a frame contains two or more images, the PLTE chunk in one image does not affect the display of the other, unless one image is a PND without a PLTE chunk, that has been declared by the DHDR ID field to depend on the other.

Note that a composite image consisting only of indexed-color images should not be assumed to contain 256 or fewer colors, since the individual palettes do not necessarily contain the same set of colors. Encoders can supply a gPLT chunk with a reduced global palette, to help decoders build an appropriate palette when necessary.

9 Security Considerations

Security considerations are addressed in the basic PNG specification.

An infinite or just overly long loop could give the appearance of having locked up the machine, as could an unreasonably long inter-frame delay. Therefore a decoder should always provide a simple method for users to escape out of a loop or delay, either by canceling the MNG entirely or just proceeding on to the next SEEK chunk.

The gPLT chunk contains a "name" field that might be printed or displayed as text by some applications. As with the tEXt chunk, any non-printable characters in the gPLT "name" field, especially the ESC character, should not be displayed directly.

No known additional security concerns are raised by this format.

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

10 Appendix: Examples

10.1 Example 1: Simple movie

```
\211 M N G \r \n ^z \n  # MNG signature
MHDR 720 468 720 468    # width and height
     20 65536  # 20 frames, max chunk length = 65kbytes
     30  3 60  # 10 frames per second, duration 60 ticks
     77000  8  # default gamma is 0.77, max_bit_depth 8
     3         # max 3 samples per pixel (color type 2)
     0         # not OK discard
     000000    # Six reserved bytes
tEXtTitle\0Sample Movie

SEEK 0 n1
IHDR 720 468 8 2 0 0 0   # DEFI 0 is implied
IDAT ...
IEND
```

```
DHDR 0 1 1 20 30 100 220 0   # A PNG-delta frame
IDAT ...       # The IDAT gives the 20x30 block
DEND           # of deltas

DHDR 0 1 1 20 30 102 222 0     # Another PNG-delta frame
IDAT ...       # This time the deltas are in a 20 x 30
DEND           # block at a slightly different location

SEEK n1 n2      # Ok to restart here because a
                # complete PNG frame follows
IHDR 720 468 ...
IDAT ...
IEND

DHDR 0 1 1 720 468 0 0 0    # Another PNG-delta frame
IDAT ...   # The entire 720x468 rectangle changes
DEND       # this time.

SEEK n2 0
MEND            # end of MNG datastream
```

10.2 Example 2: Single composite frame

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

```
\211 M N G \r \n ^z \n # MNG signature
MHDR 1024 512 500 500 # width, height
     1 8192   # nframes, maxchunklen
     1 0 0    # frame duration can be zero since there's
              # only one frame but ticklength must be nonzero
     100000   # default gamma value is 100000 (gamma=1.0)
     16 1     # max depth 16, max 1 sample per pixel
     0        # not ok to discard
     000000   # Six reserved bytes
BACK 0 64 64 192     # sky blue background
FRAM                 # composite frame, 1024 x 512
LOCA 0 6 6           # Location of first image
                     # DEFI 0 is implied
IHDR 500 500 16 0 .. # A 16-bit graylevel image
gAMA 50000           # gAMA chunk takes precedence over
IDAT ...             # the default gamma value
IEND                 # End of image
                     # SHOW 0 is implied
```

```
LOCA 0 518 6          # Location to display a modified image.
DHDR 0 1 3 0 0 0 0 0 # reload image 0 and modify it
COPY gAMA tEXt faLS IDAT  # establish chunk order
tEXtComment\0The faLS chunk is described in ftp://swrinde....
faLS ...              # Apply pseudocolor to basis image
DEND                  # End of image

LOCA 0 900 400        # Overlay near lower right-hand corner
IHDR 101 101 2 3 ... # Image 0 is redefined, but this does
                      # not affect the images already on screen
gAMA 50000            # We need a new gAMA because
PLTE ...              #    this is not a PND datastream
tRNS ...              # It's transparent (maybe a logo)
IDAT ...              # Note that the color type can differ
IDAT ...              #    from that of the other images.
IEND                  # End of image

ENDF                  # End of composite frame
MEND                  # End of MNG datastream
```

10.3 Example 3: Movie with sprites

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

```
\211 M N G \r \n ^z \n  # MNG signature
MHDR 512 512 512 512  # Start of MNG datastream
     0 0           # nframes, maxchunklen undefined
     30 3 3000    # 10 frames/sec, not more than 100 sec
     50000 8      # default_gamma 0.5, max_bit_depth 8
     1            # max samples per pixel
     0            # not ok to discard
     000000       # Six reserved bytes
FRAM              # First frame
DEFI 1            # Define image 1
                  # Location for image 1 is (0,0)
IHDR 512 512 ... # it's a full-display PNG image
etc               # chunks according to PNG spec
IEND              # SHOW 1 is implied by DEFI 1

DEFI 2            # Define image 2
LOCA 300 200      # Location for image 2
IHDR 32 32 ...    # It's a small PNG
gAMA 50000
IDAT ...
DEND              # IEND is omitted
ENDF              # end of frame
```

```
FRAM             # Second frame
                 # New location for image 1 is still 0,0
SHOW 1           # Display image 1 from previous frame
LOCA 1 10  5     # New (delta) location for image 2
SHOW 2           # Retrieve image 2 from previous frame,
CLON 2 3         # make another copy of it as image 3
LOCA 0 400 500   # Location for image 3
DHDR 3 1 3 0 0 0 0 0     # Modify image 3
tRNS ...         # Make it semitransparent
DEND
ENDF             # End of second frame

FRAM             # Next frame (repeat this FRAM-ENDF
                 #   sequence with different locations to
                 #   move the images around)
                 # New location for image 1 is still 0,0
SHOW 1           # Display image 1 from previous frame
LOCA 1 10 5      # New (delta) location for image 2
SHOW 2           # Retrieve image 2 from previous frame,
LOCA 1 5 -2      # New location for image 3
SHOW 3           # Modify image 3
ENDF               # End of frame

FRAM           #  Another frame
etc.
ENDF
etc.               # More frames
MEND               # End of MNG datastream
```

10.4 Example 4: “Fading in” a transparent image

The opaque parts of this image will “fade in” gradually. This technique won’t work with color_type 4 or 6 images that have partially transparent pixels. You would use a series of PND datastreams with IDAT chunks, instead, that add the desired amount to each individual alpha sample but have zeroes in the color component deltas. You can, however, “fade out” such images with the `fADE` chunk.

```
\211 M N G \r \n ^z \n # MNG signature
MHDR 64 64 64 64  # width, height
     16 8192      # nframes, maxchunklen
     30 6 140  # tick length, frame and total duration
     50000     # default gamma
     8 1       # max bit depth 8, 1 sample per pixel
     0         # not ok to discard
     000000    # Six reserved bytes
BACK 1 192 192 192 # "browser gray" default background
DEFI 1
IHDR ...    # PNG header
PLTE ...
tRNS ...    # Entries are zero for the transparent color
            # and 16 for the nontransparent ones.  They
            # will be barely visible)
IDAT ...
IEND
LOOP 0 0 15
DHDR 1 1 3 0 0 0 0 1
fADE 2 16   # Add 16 to alpha for all nontransparent
DEND        # colors.
ENDL 0      # Repeat loop.  After 15 iterations, the
            # opaque colors will end up with alpha=255
            # and the transparent ones will still be 0.
dURA 60     # Hold the last frame for at least 60 ticks
SHOW 1      # (2 sec).  Applications might show it longer,
            # or they might ignore dURA; it's ancillary.
MEND        # end of MNG
```

10.5 Example 5: Storing three-dimensional images

In this example, we store a series of twenty-four 150 x 150 x 150 blocks of 8-bit voxels. Each block is stored as a composite frame with the first image being a PNG whose pixels represent the top layer of voxels, which is followed by 149 PND images representing the rest of the layers of voxels. Only one `image_id` is defined, through which the basis image is passed along from PNG to PND to PND. This example also illustrates the use of unregistered ancillary chunks that describe the x, y, and z scales and pixel calibration.

```
\211 M N G \r \n ^z \n # MNG signature
MHDR 150 150 150 150 24  # width, height, nframes
     65000       #  maxchunklen (doesn't have to be 2^n)
     30 0 0      # tick length, duration (can be zero)
     100000      # default gamma (voxel data is linear)
     16 1        # max bit depth 8, 1 sample per pixel
     0           # not ok to discard
     000000      # Six reserved bytes
```

```
tEXtTitle\0Weather modeling results
tEXtComment\0The pcAL, xsCL, ysCL, zsCL, and tsCL chunks
 in this file are written according to the PNG Sci-vis
 chunks specification version 0.960816 available at
 ftp://swrinde.nde.swri.edu/pub/png-group/documents/

xsCLkilometers\0 0\0 150 # sci-vis "xsCL" chunk
ysCLkilometers\0 0\0 150 # sci-vis "ysCL" chunk
zsCLHeight (kilometers)\0 0\0 15
tsCLTime (hours)\0 0\0 24 # see proposed sci-vis chunks
pcAL 0 2 Degrees Celsius\0 0\0 45  # sci-vis "pcAL" chunk
SAVE
SEEK 0 3588720    # 3588720 bytes to the next SEEK chunk

FRAM             # initial composite image
IHDR 150 150 16  # width, height, bit depth for top layer
     0 0 0 0     # color, comp, filter, interlace
IDAT ...
IEND             # no DEFI chunk, so it's image 0

DHDR 0 1 0       # source=0, PNG, pixel subtraction,
     150 150 0 0 0  # block is entire image, copysafe
IDAT ...     # IHDR is omitted; everything matches top
DEND         # IEND is also omitted

etc.   # repeat DHDR through DEND 148 more times
ENDF   # end of first block
SEEK 3588720 4621885

etc.   # Repeat SEEK through ENDF 19 more times

SEEK 2285321 0

MEND   # end of MNG
```

10.6 Example 6: Tiling

Here's another composite frame, illustrating the use of the LOOP syntax to tile a large (1024 by 768) image area with a small (128 by 64) image.

```
\211 M N G \r \n ^z \n  # MNG signature
MHDR 1024 768 128 64 ... # Start of MNG datastream
FRAM
LOCA 0 0 -64  # set up an offscreen copy of the tile
DEFN 1        # give it ID == 1, don't show it
IHDR 128 64 ... # immediately
PLTE ...
IDAT ...      # Nothing will be displayed because it's
IEND ...      # outside the 1024 by 768 composite frame
              # and because we used DEFN instead of DEFI

LOOP 0 0 12   # Y loop -- make 12 rows of tiles
LOCA 1 0 64   # move the first copy down 64 rows
SHOW 1        # display it
CLON 1 2      # create a second copy of the tile

LOOP 1 0 7    # X loop - 7 additional columns
LOCA 1 0 128  # move it to the right 128 columns
SHOW 2        # use the second copy
ENDL 1

ENDL 0
ENDF
MEND
```

10.7 Example 7: Scrolling

Here is an example of scrolling a 3000-line-high image (perhaps an image of some text, but could be anything) through a 256-line-high window with an alpha-blended border.

```
\211 M N G \r \n ^z \n  # MNG signature
MHDR 512 256      # width and height on screen
     512 3000     # max_image must accommodate the
                  # largest stored image
     3257        # max no of frames
     32000       #  maxchunklen
     30 1 3257   # tick length, duration, total dur.
     50000       # default gamma
     4 1         # max bit depth 4, 1 sample per pixel
     0           # not ok to discard
     000000      # Six reserved bytes
FRAM
DEFN 1            # Define image 1 but don't display now
LOCA 0 0 256      # initially it's offscreen, just
                  # below the 512 by 256 window
IHDR 512 3000 ... # A PNG datastream containing the
PLTE ...          # text (or whatever) to be scrolled
IDAT ...
IEND

DEFI 2
IHDR 512 256 ... # A PNG datastream containing some kind
PLTE ...         # of alpha-blended border that is
tRNS ...         # transparent in the center
IDAT ...
IEND

ENDF

LOOP 0 0 3256
FRAM
LOCA 1 0 1 # Jack image 1 up one scanline, 3256 times
SHOW 1     # It ends up just above the 512 by 256 window
           # The border does not move (LOCA 0 0 0 implied)
SHOW 2     # Overlay the transparent border
ENDF
ENDL 0

MEND
```

11 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.berkeley.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@w3.org
- Dave Martindale, davem@cs.ubc.ca
- Carl Morris, msftrncs@htcnet.com
- Owen Mortensen, ojm@csi.compuserve.com
- Keith S. Pickens, ksp@swri.edu
- Glenn Randers-Pehrson, glennrp@arl.mil or randeg@alumni.rpi.edu
- Greg Roelofs, newt@pobox.com
- 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

Trademarks:

- Macintosh is a trademark of Apple Computer, Inc.
- Windows 3.1 is a trademark of Microsoft Corporation.

Author's Address

Glenn Randers-Pehrson
U.S. Army Research Laboratory
ATTN: AMSRL-WM-TD
Aberdeen Proving Ground, MD 21005-5066

Phone: (410) 278-6554

EMail: glennrp@arl.mil or randeg@alumni.rpi.edu

End of MNG Specification. Expires 07 March 1997