PNG-GROUP-DRAFT
MNG
Expires: 28 Feb 1997

PNG Development Group
mpng-list@dworkin.wustl.edu
28 Aug 1996

MNG (Multiple Network Graphics) Format Version 0.960828

File

draft-mng-960828.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 twelfth MNG draft (draft-mng-960824)

- Removed the composite frame specification and defined FRAM/ENDF chunks. Moved the c-frame chunks up to top-level MNG.
- changed SODE to SAVE/SHOW/DISC and added ID field to DHDR
- Changed lOOP/eNDL to LOOP/ENDL and added LOOP-related security info.
- added BLOC chunk and allow IDAT to represent block-deltas
- Removed the dISC chunk because its function is better done explicitly with DISC
- Added SYNC chunk and removed the sync function from the SEEK chunk.
- Removed references to PHDR and PEND and stated that the spec may be extended at some future date to include the PNP format.

Abstract

This document presents the *[proposed]* the format of a MNG (Portable Network graphics Frames) stream. MNG is a multiple-image format that can contain simple animations (slide shows) comprised of PNG single-image streams. It can also incorporate PND (Portable Network Delta) streams, which are defined herein,

A PND stream 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 stream for each.

The MNG format provides a mechanism for using a subimage as a "sprite" that can be moved from one location to another in subsequent frames, without having to repeat the subimage pixel data.

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 stream.

The MNG and PND formats use the same chunk structure that is defined in the PNG specification, and share other features of the PNG format.

Contents

1 Introduction 2
2 MNG chunks 3
2.1 Critical MNG chunks 3
2.1.1 MHDR MNG stream header 3
2.1.2 MEND End of MNG stream 5
2.1.3 LOCA Subimage location 5
2.1.4 IHDR, PNG chunks, IEND 5
2.1.5 DHDR, PND chunks, DEND 6
2.1.6 FRAM Frame header 6
2.1.7 ENDF End of frame 6
2.1.8 SAVE Save a subimage for reuse 6
2.1.9 SHOW Show a subimage 7
2.1.10 DISC Discard a subimage 7
2.1.11 SEEK Seek point 7
2.1.12 LOOP Begin a loop 8
2.1.13 ENDL End a loop 9
2.1.14 SYNC Synchronize display 9
2.2 Ancillary MNG chunks 9
2.2.1 dURA Duration 9
2.2.2 gPLT Global Palette 9
2.2.3 pSIZ Physical Image Size 11
2.2.4 tEXt, zTXt, tIME Text, Time chunks 11

3 The PND format . . . 11
3.1 PND critical chunks . . . 12
3.1.1 DHDR PND stream header . . . 12
3.1.2 BLOC location of a delta block . . . 13
3.1.3 DEND End of PND stream . . . 13
3.1.4 PROM Promotion of basis image . . . 13
3.1.5 IHDR, PNG chunks, IEND . . . 14
Image replacement . . . 14
Image pixel deltas . . . 15
Image alpha deltas . . . 15
No change to pixel data . . . 15
3.1.6 COPY Chunks to be copied . . . 15
3.1.7 NOCO Chunks not to be copied . . . 16
3.2 PND ancillary chunks . . . 17
3.2.1 fADE fade in or out . . . 17
3.2.2 tEXt, zTXt, tIME Text, Time chunks . . . 17
4 Retaining image data . . . 17
5 Decoder handling of fatal errors . . . 18
6 Decoder handling of interlaced files . . . 18
7 Decoder handling of palettes . . . 18
8 Security Considerations . . . 19
9 Appendix: Examples . . . 19
9.1 Example 1: Simple movie . . . 19
9.2 Example 2: Single composite frame . . . 20
9.3 Example 3: Movie with sprites . . . 21
9.4 Example 4: “Fading in” a transparent image . . . 22
9.5 Example 5: Storing three-dimensional images . . . 23
9.6 Example 6: Tiling . . . 24
10 Credits . . . 25

1 Introduction

This *[proposed]* specification defines the format of the MNG (Multiple Network Graphics) stream. It also defines the format of the PND (Portable Network graphics Delta) stream, which can be a component of a MNG stream.

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

MNG is pronounced "Ming."

When the MNG stream is a standalone file, the first eight bytes 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 stream backwards. These capabilities may 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 may 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 stream consists of a `MHDR` chunk, followed by one or more frame definitions, followed by the `MEND` chunk. The first frame must be a PNG stream (`IHDR`, PNG chunks, `IEND`) or a group of image definitions (including at least one PNG stream) enclosed in a `FRAM/ENDF` pair.

Subsequent images can be a PNG stream or a PND stream (`DHDR`, PND chunks, `DEND`), or groups of image definitions enclosed in `FRAM/ENDF` pairs. Each chunk of the MNG stream or of any image definition is an independent entity, i.e., no chunk is enclosed in the data segment of another chunk.

An independent PNG stream, with a PNG signature, is also a valid MNG stream. This kind of MNG stream must contain only a single image.

2 MNG chunks

This section describes chunks that can appear at the top level of a MNG stream. 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 stream header

```
4 bytes: max_frame_width  (unsigned nonzero integer)

4 bytes: max_frame_height (unsigned nonzero integer)
```

```
4 bytes: max_number_of_frames (unsigned nonzero integer)
         There are not more than max_number_of_frames
         in this MNG stream.

4 bytes: max_chunk_length (unsigned nonzero integer)
         No chunk in this stream, including in any
         included PNGs, has a data field
         exceeding this length.

4 bytes: ticks_per_second  (unsigned nonzero integer)

4 bytes: frame_duration (unsigned integer) in
         ticks.  The time 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
         stream.  The sum of the individual frame_durations
         must not exceed this value (under actual playback
         conditions the display is likely to take longer).

4 bytes: default_gamma (unsigned integer)
         The value of gamma, times 100000, to be assumed
         for any images or subimages in the stream 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)

1 byte:  background_source (unsigned integer)
         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.
         3: use the first image in the stream as the
            background for all subsequent images.  When
            background_source == 3, the first image must
            must not be transparent and its width and height
            must be max_frame_width and max_frame_height.

2 bytes: red_background (unsigned integer)
```

```
2 bytes: green_background (unsigned integer)

2 bytes: blue_background (unsigned integer)

1 byte:  ok_to_discard (unsigned integer)
         0: image data of each frame must be retained
            until the subsequent frame is processed.
         1: image data may be discarded after processing
            each frame.  This is a promise that this MNG
            stream contains no PND streams. If
            background_source == 3, the image data from
            the first image must be retained even if
            ok_to_discard == 1.
```

[Should we reserve a byte for requesting audio resources?]

2.1.2 MEND End of MNG stream

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

2.1.3 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:

```
1 byte:  Loca_Delta_type
         0: LOCA data gives X and Y directly
         1: LOCA positions are determined by adding
            the LOCA data to the position of the
            basis subimage
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 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.

If there is no basis subimage (no DHDR or SHOW chunk appears), the position is computed with the basis position being (0,0). If there is a basis subimage but no LOCA chunk, the new subimage is displayed at the same location as the basis subimage.

2.1.4 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 MHDR chunk. After applying the LOCA offset, the image must be clipped to fit the limits from the MHDR or FRAM chunk, whichever is smaller.

2.1.5 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 format. The image width and height must not exceed max_frame_width and max_frame_height from the MHDR chunk. After applying the LOCA offset, the image must be clipped to fit the limits from the MHDR or FRAM chunk, whichever is smaller.

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

2.1.6 FRAM Frame header

Beginning of a composite frame. A composite frame is made up of one or more PNG or PND streams.

```
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 MHDR 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 MHDR chunk)
```

The LOCA chunk can be used to specify the placement of each subimage within the composite frame.

For every FRAM chunk, there must be a corresponding ENDF chunk later in the MNG stream.

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 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.

2.1.7 ENDF End of frame

End of composite frame. This chunk has a zero data length. Every ENDF chunk, there must be a corresponding FRAM chunk earlier in the MNG stream.

2.1.8 SAVE Save a subimage for reuse

```
2 bytes: subimage_id (unsigned integer) subimage identifier
         to be given to the subimage that immediately
         follows the SAVE chunk.  Subsequent DHDR, SHOW,
         and DISC chunks can use this number to locate it.
```

If new_id is an identifier that already exists, the basis image previously associated with the identifier is discarded. The SAVE can be followed by another SAVE chunk, with a different value of subimage_id, to make another independent copy of the subimage.

If a IHDR-IEND sequence is not immediately followed by a SAVE chunk, then "SAVE 0" is implied, unless the "ok_to_discard" field of the MHDR chunk is set.

2.1.9 SHOW Show a subimage

Retrieve and display a previously-saved subimage, without making any changes.

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

An instance of the subimage 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 subimage.

The SHOW chunk can be followed by a SAVE chunk. In this case, two identical copies of the resulting basis image will be available for use by subsequent DHDR or SHOW chunks, with different identifiers. Viewers must only display the image once; the splitting into two basis images occurs after the display operation.

It is not necessary to follow an IHDR-IEND or DHDR-DEND sequence with a SHOW chunk. Such streams are always displayed if they are within the width and height bounds of the frame.

2.1.10 DISC Discard a subimage

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

2.1.11 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 stream where a restart is possible, and no information appearing prior to the SEEK chunk (other than the information in the MHDR 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 chunk would not be sufficient because the PNG 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 MHDR chunk.

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.

2.1.12 `LOOP` Begin a loop

The `LOOP` chunk provides a "shorthand" notation that can be used to avoid repeated identical chunks in a MNG stream. Its contents are

```
1 byte: start_loop_level: 0-255
1 byte: repeat count (unsigned integer) range 0 to 2^31 - 1
        0-2^31 -1: number of loop iterations
```

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 frame for the first frame in the loop is the frame immediately preceding the `LOOP` chunk, but in subsequent interations the last frame in the loop becomes the basis frame for the first frame of the next iteration.

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

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

2.1.13 `ENDL` End a loop

This chunk ends a loop that begins with the `LOOP` chunk.

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

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 stream.

2.1.14 `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 stream (not defined in this specification) with the display, to synchronize stereo images, and the like.

2.2 Ancillary MNG chunks

2.2.1 dURA Duration

Duration of display.

```
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. 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 may 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 may be used to construct a reduced palette to which the truecolor image may 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 may permit applications

or people to choose the appropriate one when more than one suggested palette appears in a MNG stream. The "name" string must consist only of printable ASCII characters and may not have leading or trailing blanks, but may have single embedded blanks. There must be at least one and no more than 79 characters in the name. Names are case-sensitive. 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.

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 a subimage 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 within a composite frame description. If present, they must appear prior to any `IHDR` or `DHDR` chunk in the composite frame. The `gPLT` chunk may appear for any color type. When a subimage contains a `PLTE` or `sPLT` suggested palette, the `gPLT` data takes precedence.

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 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),
         corresponding to composite_frame_height.
```

The `pSIZ` chunk must appear prior to the `FRAM` chunks to which it pertains.

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 subimage 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 stream describes a single image, by giving the changes from a previous PNG (Portable Network Graphics) or another PND image. While it is possible for a PND stream to be stored in a standalone file (where the basis image is available in another standalone file), a PND stream will normally be found as a component of a MNG stream. The PND format may be extended at some later date to include a PNP (Portable Network Photo) stream.

If the PND stream is a standalone file (not usually the case), the first eight bytes are

```
140 80 78 68 13 10 26 10
```

which is similar to the PNG signature with "\213 P N D" instead of "\211 P N G" in bytes 1-4. Use ".pnd" as the file suffix.

PND is pronounced "Ping-delta."

The decoder must have available a basis (decoded) image from which the original chunk data is known. How the basis 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 MNG datastream. The basis image can be the result of decoding a PNG or another PND stream. If alpha or transparency is present, then the decoder must also have available the original background against which the basis image was displayed, if transparency or alpha are to be composited against the background.

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 stream 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 stream header

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

1 byte: image_type
  0: Image type is unspecified.  An IHDR chunk
     must be present.
  1: Image type is PNG.   IHDR may 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.
```

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.

```
1 byte: delta_type
  0: image replacement
  1: pixel subtraction, by bytes, modulo 256.
  2: alpha subtraction, by bytes, modulo 256.  The
     IHDR - IEND data are written as a grayscale
     image (color type 0) but the decoded samples
     are to be 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
```

[We might want to add other delta_types like an interplane Paeth filter]

```
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 NOCO 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 BLOC location of a delta block

This chunk gives the location of the block of pixels that are to be modified by the subsequent IHDR/IDAT chunks in this PND stream or in a subsequent PND for which this PND serves as the basis.

```
4 bytes: block_x_location (unsigned integer)
4 bytes: block_y_location (unsigned integer)
```

block_x_location and block_y_location are measured in pixels, downward and to the right of the top left corner of the image.

3.1.3 DEND End of PND stream

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

3.1.4 PROM Promotion of basis image

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

The PROM chunk must appear ahead of IHDR.

3.1.5 IHDR, PNG chunks, IEND

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

See the PNG specification for the format of the PNG chunks. The PNG stream must contain at least IHDR and IEND but may 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 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 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 and the image_type is 1 and delta_type is 0, 1, or 3 in the DHDR chunk, and if 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 stream as though the IHDR were present, immediately after the DHDR chunk, with all IHDR chunk data identical to that of the basis image.

When the IHDR chunk is present and its width and height do not match those of the basis image, and delta_type is 1 or 2, then only the pixels within the rectangle having the specified width and height and located within the basis image by means of the BLOC chunk are modified. The BLOC chunk must appear in the PND stream or in a prior basis image upon which the stream depends.

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, all unknown chunks appearing in the partial PNG stream 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 IDAT chunk is not needed except that unknown chunks appear that have ordering requirements with respect to IDAT, an empty IDAT chunk can be included to serve as a place marker. If an instance of PLTE is needed, then a complete PLTE chunk must appear. (A zero-length PLTE chunk must not be used for this purpose because that would be a legal PNG chunk describing a palette with no entries).

The IEND chunk can be omitted, if it would be the last chunk in the PND stream 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 stream or from a sequence of PND streams that depend upon a PNG stream.

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 previous 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 bytes in the IDAT chunk by subtracting the basis image bytes from the new image bytes, modulo 256. When decoding the IDAT chunk, the new image bytes are obtained by adding the delta bytes to the basis 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.

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 updated alpha samples are calculated in the same manner as the updated pixels 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 need match that of the basis image.

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

3.1.6 `COPY` Chunks to be copied

Chunks to be copied

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

`COPY` 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.

There can be multiple instances of the `COPY` chunk. `COPY` chunks must appear before `IHDR` if it is present.

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 basis image, that obeys the chunk-ordering rules for copy-safe chunks given in the PNG specification.

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 streams, if there are no changes.

If a chunkname appears in a `COPY` chunk and also appears in a `NOCO` chunk, the last instance takes precedence.

Applications that reconstruct a PNG stream from a PND stream and a basis PNG or PND stream should not write the `COPY` chunk to the resulting PNG stream, 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.

3.1.7 NOCO Chunks not to be copied

Chunks not to be copied

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

NOCO 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.

There can be multiple instances of the NOCO chunk. The NOCO chunks must appear before IHDR if it is present.

If a chunkname appears in a COPY chunk and also appears in a NOCO chunk, the last instance takes precedence.

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 stream from a PND stream and a basis PNG or PND stream should not write the NOCO chunk to the resulting PNG stream.

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 streams, 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 Retaining image data

When dependent frames (frames containing PND streams) are present, the decoder must retain information about the previous frame for use in decoding the dependent frame. It is never necessary to retain information from frames earlier than the previous frame, except for the original background, which must be retained for the entire duration of the MNG stream if it is required.

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.

The LOCA and BLOC data associated with an image must also be retained, in addition to its internal PNG chunk data.

5 Decoder handling of fatal errors

When a fatal error is encountered, such as an unknown critical chunk in an image or subimage, 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.

6 Decoder handling of interlaced files

Decoders are required to be able to interpret streams 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 streams 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.

7 Decoder handling of palettes

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

If PLTE is present in a PND stream, 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 subimages, the PLTE chunk in one subimage does not affect the display of the other, unless one subimage 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.

8 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.

9 Appendix: Examples

9.1 Example 1: Simple movie

```
\211 M N G \r \n ^z \n  # MNG signature
MHDR 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
     2 0 0 0 0 # Background not required, not OK discard
tEXtTitle\0Sample Movie

SEEK 0 n1
IHDR 720 468 ...
IDAT ...
IEND        # SAVE 0 is implied


DHDR 0 1 1 0  # A PNG-delta frame
IDAT ...
DEND

DHDR 0 1 1 0  # Another PNG-delta frame
IDAT ...
DEND

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

DHDR 0 1 1 0    # another PNG-delta frame
IDAT ...
DEND

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

9.2 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 1 8192 # width, height, 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       # depth 16
     1 64 64 192  # sky blue background
     1        # ok to discard
FRAM 1024 512        # composite frame, 1024 x 512
LOCA 0 6 6           # Location of first subimage
IHDR 500 500 16 0 .. # A 16-bit graylevel image
gAMA 50000
IDAT ...
IEND                 # End of subimage
SAVE 1               # keep subimage for later use

LOCA 0 518 6         # Location for second subimage.
DHDR 1 1 0 1         # reload object 1 and modify it
IHDR 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 IDAT; pixels don't change
IEND
DEND                 # End of subimage
DISC 1               # We have no further need of this as
                     # a basis image for subsequent PNDs

LOCA 0 900 400       # Overlay near lower right-hand corner
IHDR 101 101 2 3 ...
gAMA 50000           # We need a new gAMA 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

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

9.3 Example 3: Movie with sprites

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

```
\211 M N G \r \n ^z \n  # MNG signature
MHDR 512 512     # Start of MNG stream
     30 3 3000   # 10 frames/sec, not more than 100 sec
     50000 8     # default_gamma 0.5, max_bit_depth 8
     2 0 0 0     # don't need background
     0           # not ok to discard
FRAM 512 512     # First frame
LOCA 0 0         # Location for object 1
IHDR 512 512 ... # it's a full-display PNG image
etc              # chunks according to PNG spec
IEND
SAVE 1           # keep previous subimage as object 1
LOCA 300 200     # Location for object 2
IHDR 32 32 ...   # It's a small PNG
gAMA 50000
IDAT ...
IEND
SAVE 2           # Save object 2 and export to next frame
ENDF             # end of frame

FRAM 512 512   # Next frame (repeat this CHDR-CEND
                 #   sequence with different locations to
                #   move the objects around)
                # New location for object 1 is still 0,0
SHOW 1           # Display object 1 from previous frame
LOCA 1 10   5     # New (delta) location for object 2
SHOW 2           # Retrieve object 2 from previous frame,
SAVE 3           # make another copy of it as object 3
LOCA 0 400 500   # Location for object 3
DHDR 3 1 0 2     # Modify object 3
tRNS ...         # Make it semitransparent
DEND
ENDF               # End of frame

FRAM 512 512   #  Another frame
etc.
ENDF
etc.               # More frames
MEND               # End of MNG stream
```

9.4 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 streams 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 16  # width, height, nframes
     8192      #  maxchunklen
     30 6 140  # tick length, frame and total duration
     50000     # default gamma
     8         # max bit depth 8
     0         # use application supplied background
   192 192 192 # "browser gray" default background
     0         # not ok to discard
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
SAVE 1
LOOP 0 15
DHDR 1 1 0 0
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
```

9.5 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 subimage being a PNG whose pixels represent the top layer of voxels, which is followed by 149 PND subimages representing the rest of the layers of voxels. Only one "object" is saved, 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 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)
     8           # max bit depth 8
     2 0 0 0     # background not needed
     0           # not ok to discard
```

```
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/

SEEK 0 3588720    # 3588720 bytes to the next SEEK chunk
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

FRAM 150 150   # initial composite image
IHDR 150 150 8   # width, height, bit depth for top layer
     0 0 0 0     # color, comp, filter, interlace
IDAT ...
IEND           # no SAVE chunk, so it's object 0

DHDR 0 1 0 0 # PNG pixel subtraction, copysafe, source=0
            # IHDR is omitted; everything matches top
IDAT ...    # no need to repeat pcAL, xsCL and ysCL
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
       # Must repeat the x, y, zSCL chunks because all
       # data is forgotten when SEEK is encountered

SEEK 2285321 0

MEND   # end of MNG
```

9.6 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 ...   # Start of MNG stream
FRAM 1024 768
LOCA 0 0 -64  # set up an offscreen copy of the tile
IHDR 128 64 ...
PLTE ...
IDAT ...      # Nothing will be displayed because it's
IEND ...      # outside the 1024 by 768 composite frame
SAVE 1        # give it ID == 1

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

LOOP 1 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 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
- 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

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 28 February 1997