PNG-GROUP-DRAFT
MNG
Expires: 01 May 1997

PNG Development Group
mpng-list@dworkin.wustl.edu
01 Nov 1996

MNG (Multiple Network Graphics) Format Version 19961101

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 twenty-first MNG draft (draft-mng-19961025)

- Restore “LOCA” chunk and eliminate “PROP” chunk.
- Add visibility byte to the show chunk.
- Added framing_modes to accomodate sprite and full-sprite modes.
- Add CLON mode 2 (renames image_id)
- Eliminated SVCF chunk.
- Rewrote Example 8, not using SVCF.
- Reorganized the order of presentation of MNG critical chunks.
- Rewrite of DHDR, requiring certain fields to be omitted when delta_type==0 or delta_type==5.
- An empty SBYK chunk clears the SBYK list for an image_id.
- SBYK no longer depends on INHR rules.
- Fixed some typos.

Abstract

This document presents the *[proposed]* 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 shares 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, storage of voxel data, and converting GIF animations to MNG format.

Contents

1 Introduction 2
2 MNG chunks 3
2.1 Critical MNG control chunks 3
2.1.1 `MHDR` MNG datastream header 3
2.1.2 `MEND` End of MNG datastream 5
2.1.3 `NEED` Resources needed 6
2.1.4 `SAVE` Save decoder state 6
2.1.5 `SEEK` Seek point 7
2.1.6 `LOOP, ENDL` Define a loop 7
2.2 Critical MNG image defining chunks 9
2.2.1 `DEFI, DEFN` Define an image 9
2.2.2 `IHDR`, PNG chunks, `IEND` 9
2.2.3 `BASI`, PNG chunks, `IEND` 9
2.2.4 `DHDR`, PND chunks, `DEND` 10
2.2.5 `CLON` Clone an image 10
2.2.6 `DISC` Discard images 11
2.2.7 `BLNK` Define a blank image 12

2.2.8 INHR Global INHR chunk . . . 12
2.2.9 SBYK Global SBYK chunk . . . 12
2.3 Critical MNG image displaying chunks . . . 13
2.3.1 LOCA Image location . . . 13
2.3.2 BACK Background . . . 14
2.3.3 CLIP Image clipping boundaries . . . 14
2.3.4 FRAM Delimit a frame . . . 15
2.3.5 SHOW Show images . . . 19
2.4 Ancillary MNG chunks . . . 20
2.4.1 sPLT Suggested palette . . . 20
2.4.2 fPRi Frame priority . . . 22
2.4.3 PNG ancillary chunks . . . 23
3 The PND format . . . 23
3.1 PND critical chunks . . . 24
3.1.1 DHDR PND datastream header . . . 24
Entire image replacement . . . 26
Block pixel addition . . . 26
Block alpha addition . . . 26
Block pixel replacement . . . 27
Block alpha replacement . . . 27
No change to pixel data . . . 27
3.1.2 DEND End of PND datastream . . . 27
3.1.3 PROM Promotion of basis image . . . 27
3.1.4 IHDR/IPNG, PNG chunks, IEND . . . 28
3.1.5 INHR Inherited chunks . . . 29
3.1.6 SBYK Select chunks by keyword . . . 31
3.2 PND ancillary chunks . . . 32
3.2.1 fADE Fade in or out . . . 32
3.2.2 PNG chunks . . . 33
4 Retaining image data . . . 33
5 Decoder handling of fatal errors . . . 33
6 Decoder handling of interlaced files . . . 34
7 Decoder handling of palettes . . . 34
8 Security considerations . . . 34
9 Appendix: Examples . . . 35
9.1 Example 1: Simple movie . . . 35
9.2 Example 2: Single composite frame . . . 36
9.3 Example 3: Movie with sprites . . . 37
9.4 Example 4: “Fading in” a transparent image . . . 38
9.5 Example 5: Storing three-dimensional images . . . 39
9.6 Example 6: Tiling . . . 40
9.7 Example 7: Scrolling . . . 41
9.8 Example 8: Converting a GIF animation to MNG . . . 41

10 Credits 43

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 defined 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
```

(decimal) 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 must 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) delimited by "`FRAM`" chunks.

While MNG datastreams are written so decoders can display frames properly by showing their subimages in the order they appear in the datastream, MNG also provides "level" information that allows viewers to composite the frames in a "sprite mode" that can offer faster composition and efficient use of display hardware.

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 delimited by "`FRAM`" chunks.

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 will 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 control chunks

This section describes critical MNG control chunks. MNG-compliant decoders must recognize and process them.

2.1.1 MHDR MNG datastream header

The "MHDR" chunk is always first in all MNG datastreams except for those that consist of a PNG datastream with a PNG signature. The "MHDR" chunk contains exactly 64 bytes:

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

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

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

4 bytes: max_stored_image_height (unsigned 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
         generated by this MNG datastream, after any
         loops are unrolled.  If this field is zero,
         max_number_of_frames is undefined.

4 bytes: max_chunk_length (unsigned 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.

8 bytes: max_bytes_of_image_storage (unsigned 64-bit integer). The maximum amount of storage required to hold all of the images that are active at any one time, assuming that the images are stored as uncompressed data, of the appropriate bit depth and color type, with scanlines padded to end on byte boundaries. This does not include the extra storage needed to carry the LOCA data for each stored image. If this field is zero, the maximum amount of image storage 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 or implicitly discarded with the SEEK chunk.

```
          1: image data can be discarded after processing
             each image.  This is a promise that this MNG
             datastream contains no PND datastreams or
             BASI, DEFI, DEFN, SHOW, or CLON chunks.

    2  bytes: max_image_id (unsigned integer).  No image
             in this MNG datastream has an image_id
             greater than this number.

    15 bytes: Reserved, must be zero.
```

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 NEED Resources needed

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

The "NEED" chunk contains a list of keywords that the decoder must recognize. Keywords are typically chunk names.

```
n bytes: keyword
1  byte: null separator
etc.
```

The "NEED" chunk should be placed early in the MNG datastream, preferably immediately after the "MHDR" chunk.

The keywords are typically 4-character chunk names, but they could be any string that a decoder is required to recognize. Each keyword string must follow the format of a "tEXt" keyword: It must consist only of printable Latin-1 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 keyword. Keywords are case-sensitive.

For example, to indicate that the file is written in accordance with this draft version of MNG, you can use

```
NEED
draft 22
```

Decoders not recognizing a chunk name or keyword in the list should abandon the MNG datastream or request user intervention. The normal security precautions should be taken when displaying the keywords.

2.1.4 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. The "SAVE" chunk must be present when the "SEEK" chunk is present.

Only one instance of the "SAVE" chunk is permitted in a MNG datastream. It is not allowed anywhere after the first "SEEK" chunk.

When an "image_id" is defined prior to the "SAVE" chunk, it is not permitted to modify the image or any of the data associated with that "image_id" after the "SAVE" chunk. Such an image can be discarded by means of the "DISC" chunk; if so, its "image_id" must not be reused. Such an image can be the subject of a "CLON" chunk. If the clone is a partial clone, it is not permitted to modify it, because this would also modify the original image..

2.1.5 SEEK Seek point

The "SEEK" chunk marks positions in the MNG datastream where a restart is possible.

```
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.
```

Applications must not use any information preceding the "SEEK" chunk, except for

- data appearing in the "MHDR" chunk
- anything appearing ahead of the "SAVE" chunk.

When the "SEEK" chunk is encountered, the decoder must restore its data to the state that existed when the "SAVE" chunk was encountered.

In addition to providing a mechanism for skipping frames or backspacing over frames, the "SEEK" chunk 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.

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 must be immediately terminated.

Multiple instances of the "SEEK" chunk are permitted. The "SEEK" chunk must not appear prior to the "SAVE" chunk. The "SAVE" chunk must also be present if the "SEEK" chunk is present.

2.1.6 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
         (via the LOCA chunk) basis images.
      1: Execution of the loop might modify or relocate
         basis images, but upon completion of the loop,
         all basis images have been restored to their
         initial state and location.
      2: Execution of the loop does not modify or
         relocate any basis images.
4 bytes: repeat_count (unsigned integer), range
         [0..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 iterations 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 "start_loop_level" 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. Note that the "loop_effect" field says nothing about the appearance of the display at the end of the loop. It describes the state of the basis images at the end of each iteration of the loop. When "loop_effect==1", all iterations of the loop are identical, and a viewer could choose to store copies of the frame buffer for redisplay. This is also true when "loop_effect=2"; furthermore, if the user has escaped from the interior of the loop it is safe to resume processing with the first chunk after the "ENDL" chunk without having to skip to a "SEEK" chunk.

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..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.

2.2 Critical MNG image defining chunks

2.2.1 DEFI, DEFN Define an image

The "DEFI" chunk defines an image and immediately makes it visible. The "DEFN" chunk defines an image without making it visible.

```
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, and DISC chunks can
         use this number to identify it.
```

The "DEFI" or "DEFN" chunk must be followed by an "IHDR" or "BLNK" chunk that introduces a PNG datastream that defines the image. If "image_id" is an identifier that already exists, the basis image previously associated with the identifier is discarded.

If an "IHDR-IEND" sequence or a "BLNK" chunk is not immediately preceded by a "DEFI" or "DEFN" chunk, then "DEFI 0" is implied, unless "ok_to_discard==1" in the "MHDR" chunk. When "ok_to_discard==1" such an image is simply displayed but not stored.

2.2.2 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 limits described in the specification for the "CLIP" chunk in Paragraph 2.3.3, below.

The PNG "oFFs" and "pHYs" chunks must be ignored by MNG viewers and simply copied by MNG editors.

2.2.3 BASI, PNG chunks, IEND

The format of the "BASI" chunk is identical to that of the "IHDR" chunk.

The "BASI" introduces a datastream that contains PNG chunks, but is not necessarily a PNG datastream. It can be incomplete or it can deviate in certain ways from the PNG specification. It can serve as a basis for a PND datastream, which must supply the missing data or correct the other deviations before the image is displayed. The end of the datastream is denoted by an "IEND" chunk.

The permitted deviations from the PNG format are:

- Multiple instances of some chunks can be present even though the PNG specification allows only one. The subsequent PND that uses this as the basis image must select only one, through the "SBYK" or similar mechanism.
- The "IDAT" chunk can be omitted or there can be a single empty "IDAT" chunk. If so, the subsequent PND that uses this as the basis image must supply the "IDAT" chunk or chunks.
- The "PLTE" chunk can be omitted or incomplete even when the color_type==3. If so, the subsequent PND that uses this as the basis image must supply a complete "PLTE" chunk.

The "BASI" chunk can be used to introduce such things as a library of "faLT" chunks from which one or another can be selected for use with any single image.

A "BASI" chunk appearing in a MNG datastream must be preceded by a "DEFN" chunk that gives the "image_id" for the basis image, and can also be accompanied by a "LOCA" chunk whose values will be inherited along with the data contained in the PNG chunks.

No provision is made in this specification for storing a BASI datastream as a standalone file. A BASI datastream will normally be found as a component of a MNG datastream. Applications that need to store a BASI datastream separately should use a different file signature and filename extension, or they can wrap it in a MNG datastream consisting of the MNG signature, the "MHDR" chunk, the "BASI" datastream, and the "MEND" chunk.

2.2.4 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 limits described in the specification for the "CLIP" chunk in Paragraph 2.3.3, below.

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

2.2.5 CLON Clone an image

Create a clone (a new copy) of an image, with a new "image_id". The "CLON" chunk contains two two-byte fields:

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

2 bytes: clone_id (unsigned integer) identifier to be
         given to the clone (new copy) to serve as the
         image_id of the new image.

1  byte: clone_type (unsigned integer)
      0: full clone of LOCA data and image data
      1: partial clone; only the LOCA data is copied
         and a link is made to the image data.
      2: rename image (this is equivalent to
         "CLON old_id new_id 1" followed by
         "DISC old_id").
```

The clone is initially identical to the basis image and has the same "LOCA", "INHR", and "SBYK" data as the basis image. Subsequent "DHDR", "SHOW", and "DISC" chunks can use the "clone_id" to identify it.

Subsequent chunks can modify, show, or discard a full clone or its associated "LOCA", "INHR", or "SBYK" data without affecting the basis image, or they can modify, show, or discard the basis image or its associated data without affecting the clone.

If an image has partial clones, and the image data in a basis image or any of its partial clones is modified, the basis image and all of its partial clones are changed. Decoders must take care that when the basis image or any partial clone is discarded, the image data is not discarded until the last remaining one of them is discarded. The "INHR" and "SBYK" data is also shared, so that if they are changed in any partial clone, they are changed in the others as well. Only the "LOCA" data can be changed independently for each partial clone.

The "CLON" chunk does not imply "SHOW".

[Since a clone will almost always need an accompanying LOCA chunk, should we have loca_delta_type, x_location, and y_location fields? If so, what should happen if a LOCA chunk is also present? Apply the CLON location data first, then the LOCA data? Ignore the LOCA data?]

2.2.6 DISC Discard images

The "DISC" chunk can be used to inform the decoder that it can discard the image data associated with the associated image identifiers. Whether the decoder actually discards the data or not, it must not use it after encountering the "DISC" chunk.

The chunk contains a sequence of zero or more two-byte image 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
         discarded and the identifier can be reused by
         a DEFI chunk.
etc.
```

If the "DISC" chunk is empty, all images except those preceding the "SAVE" chunk can be discarded. Note that the appearance of a "SEEK" chunk in the datastream implies an empty "DISC" chunk.

When an image is discarded, any "LOCA", "INHR", or "SBYK" data associated with it is also discarded.

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

Discarding images that are defined prior to the "SEEK" chunk is permitted, but the "image_id" of such images must not be reused.

When the image is a partial clone or is the source of a partial clone that has not been discarded, only the "LOCA" data can be discarded. The image data must be retained until the last remaining partial clone is discarded.

2.2.7 BLNK Define a blank image

The "BLNK" 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 "BLNK" 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==3", it also generates a "PLTE" of length "2^bit_depth", filled with zeroes.

2.2.8 INHR Global INHR chunk

An instance of the PND "INHR" chunk can be placed in the top-level MNG datastream, ahead of the "SAVE" chunk, to provide a default set of chunk inheritance rules to be used when "INHR" chunk is omitted from a PND datastream and its basis image. For the format and meaning of the "INHR" chunk, see Paragraph 3.1.5, below.

2.2.9 SBYK Global SBYK chunk

Multiple instances of the PND "SBYK" chunk can be placed in the top-level MNG datastream, ahead of the "SAVE" chunk, to provide a default set of chunk "selection by keyword" rules to be used when "SBYK" chunk is omitted from a PND datastream and its basis image. For the format and meaning of the "SBYK" chunk, see Paragraph 3.1.6, below.

2.3 Critical MNG image displaying chunks

2.3.1 LOCA Image location

Location, level, and visibility for an image.

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

The chunk's contents are:

```
2 bytes: image_id (unsigned integer).   This
         also is the image_level of the image
         in a composite frame.  The image
         having the smallest value of
         image_level is on the bottom (can be
         obscured by any other image_id), while
         the image having the smallest value is
         the top image (cannot be obscured by
         any other image_id.

1  byte: loca_delta_type (unsigned integer)
      0: LOCA data gives x_location and
         y_location directly [and depth/level?]
      1: New positions are determined by adding
         the LOCA data to the position of the
         basis image
```

```
4 bytes: x_location or delta_x_location
         (signed integer)

4 bytes: y_location or delta_y_location
         (signed integer)
```

Negative values are permitted for the X and Y image position. The positive directions are downward and rightward. The "LOCA" chunk 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 determined as described in the specification for the "CLIP" chunk Paragraph 2.3.3, below.

If there is no basis image, it is an error to set "loca_delta_type==1".

If there is no "LOCA" chunk, for a particular "image_id", the image has the following properties:

```
x location := 0
y location := 0
```

2.3.2 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 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 delimited by "FRAM" chunks, against a fresh copy of the background (they might actually follow some other procedure, but the final appearance of each frame must be the same as if they had).

Multiple instances of the "BACK" chunk are permitted in a MNG datastream. One should appear before the "SAVE" chunk, if the "SAVE" chunk is present, to give the data to be restored when a "SEEK" chunk is encountered.

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.

2.3.3 CLIP Image clipping boundaries

This chunk gives the boundaries to which images must be clipped for display. It contains the following 17 bytes:

```
1  byte: clip_delta_type (unsigned integer).
      0: CLIP data gives boundary values directly
      1: CLIP boundaries are determined by adding
         the CLIP data to their previous values

4 bytes: left_cb (signed integer).  Boundary must
         be zero or greater.

4 bytes: right_cb (signed integer).  Boundary must
         be greater than left_cb and less than or
         equal to max_frame_width from the IHDR
         chunk.

4 bytes: top_cb (unsigned integer).  Boundary must
         be zero or greater.

4 bytes: bottom_cb (unsigned integer).  Boundary
         must be greater than top_cb 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_cb   := 0
right_cb  := max_frame_width
top_cb    := 0
bottom_cb := max_frame_height
```

The left and top clipping boundaries are inclusive and the right and bottom clipping boundaries are exclusive, i.e. the pixel located at {x,y} is only displayed if

```
left_cb <= x < right_cb AND top_cb <= y < bottom_cb
```

2.3.4 FRAM Delimit a frame

The "FRAM" chunk marks the frame boundary. It also can set parameters that govern how the decoder is to behave when it encounters a "FRAM" chunk.

An empty "FRAM" chunk is just a frame boundary marker. A nonempty one is a frame boundary marker, and it also changes "FRAM" parameters, either for the upcoming frame or until reset. When the "FRAM" chunk is not empty, it contains four 1-byte fields plus a variable number of optional 4-byte fields.

```
1 byte:  framing_mode.
     0:  Don't change framing_mode.
     1:  Each image is an individual frame
     2:  The group of images appearing prior to
         the next "FRAM" chunk form a "sprite mode"
         frame (any image_id appears only once).
         All images are initially invisible and
         are made visible with explicit or implied
         "SHOW" chunks.
     3:  The group of images appearing prior to
         the next "FRAM" chunk define "sprite mode"
         frames as they accumulate (any image_id
         appears only once), and a frame boundary
         occurs after each image is defined or
         appears in a "SHOW" chunk. All images are
         initially made invisible but remain
         visible after being explicitly made
         visible.
     4:  The group of images appearing prior to
         the next "FRAM" chunk is a "full sprite
         mode" frame (any image_id appears only
         once).  Images retain their visibility
         from the previous frame and are made
         visible or invisible with explicit or
         implied "SHOW" chunks.
     5:  The group of images appearing prior to
         the next "FRAM" chunk is a "draw" mode
         frame (an image_id can appear more than
         once in the frame).
     6:  The group of images appearing prior to
         the next "FRAM" chunk define "draw" mode
         frames as they accumulate (an image_id
         can appear more than once in the frame).

1 byte:  change_frame_duration.
     0:  no
     1:  yes, for the next frame only
     2:  yes, also reset default
```

```
1 byte:  change_sync_timeout.
     0:  no
     1:  yes, for the next frame only
     2:  yes, also reset default

1 byte:  change_sync_id_list.
     0:  no
     1:  yes, for this frame only
     2:  yes, also reset default list

4 bytes: frame_duration (unsigned integer) (omit
         if change_frame_duration==0).  The range
         is [0..2^31-1] ticks.

4 bytes: sync_timeout (unsigned integer) (omit if
         change_sync_timeout==0). The range is
         [0..2^31-1].  The value 2^31-1 (0x7fffffff)
         ticks represents an infinite timeout period.

4 bytes: sync_id (unsigned integer) (omit if
         change_sync_id_list==0 or if the new list is
         empty; repeat until all sync_id's have been
         listed).  The range is [0..2^31-1].
```

When the "FRAM" parameters are changed, the new parameters affect the frame that is about to be defined, not the one that is terminated by the "FRAM" chunk.

When the "sync_id" list is changed, the number of "sync_id" entries is determined by the remaining length of the chunk data, divided by four. This number can be zero, which either inactivates the existing "sync_id" list for one frame or deletes it.

When a "SEEK" chunk is encountered, all "FRAM" parameters are reset to the values they had when the "SAVE" chunk was encountered, and, if a "sync_id" list existed, it is restored.

The initial values of the FRAM parameters are

```
framing_mode := 1
frame_duration is defined in the MHDR chunk
sync_timeout := 0x7fffffff (infinite)
sync_id list 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 can be designed to operate in "draw" mode or "sprite" mode. A "draw" mode viewer simply displays images within a frame in the order that they appear (or corresponding SHOW chunks appear) in the MNG datastream. A "sprite" mode viewer accumulates images, together with their clipping boundaries, until it

needs to display a complete frame, and then composites all of the images whose "visibility" byte is 0, from top to bottom, according to the "image_id" value for each image. Images will normally appear in the datastream in the "draw" mode order, which is bottom-to-top (i.e. the background image, which has the smallest value of "image_id", appears first in the datastream).

When "framing_mode==1", each image that becomes visible is a separate frame, delimited by its "IEND" chunk (in the case of PNG images) or "DEND" chunk (in the case of PND images). Each image that is made visible by a "SHOW" chunk also constitutes a separate frame. If "LOCA" chunks appear, they must appear prior to the image definition chunks or "SHOW" chunks to which they pertain.

When "framing_mode==2", viewers are expected to display all of the images at once, if possible, or as fast as can be managed, without clearing the display and restoring the background between images. 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 next "FRAM" chunk. Each image in the frame be given a unique "image_id", in ascending order from the background image (which can be obscured by any other image) to the top image (which can not be obscured but can obscure any other image). All images are initially invisible, and are made visible by their appearance in an explicit or implied "SHOW" chunk. If "LOCA" chunks appear, they must appear before the images to which they pertain are made visible. "Sprite" mode decoders can accumulate them and display them all at once when the next "FRAM" chunk comes along. "Draw" mode decoders can display them in the order that they appear or are mentioned in "SHOW" chunks.

When "framing_mode==3", then a frame boundary occurs after each image is made visible, without clearing the display and restoring the background between frames. Each image must be assigned to a unique "image" id, in ascending order of appearance. If "LOCA" chunks appear, they must appear before the images to which they pertain are made visible. "Draw mode" decoders can simply erase the display prior to displaying the first image, and then show each image as it is defined, without erasing the display. "Sprite" mode decoders can recomposite the display after each image has been made visible.

When "framing_mode==4", then the images are to be displayed all at once, when the frame has been completely composited, like "framing_mode==2", except that images retain their visibility from the previous frame. The frames can be made visible or invisible in any order, and "LOCA" chunks can be placed anywhere. "Sprite mode" decoders composite them in the same manner as if "framing_mode==2". "Draw mode" decoders must wait until the frame has been fully defined, and then can draw the visible images in bottom-to-top order.

When "framing_mode==5", then "draw mode" decoders can treat the frame in the same way as when "framing_mode==2". If "LOCA" chunks appear, they must appear before the images to which they pertain are made visible. A "sprite mode" decoders will have to revert to "draw mode" to display this frame type, or or else it will have to make its own clones and reassign them to unique, properly ordered unused levels. Encoders are encouraged to use "framing_mode==2" instead, unless the frame contains such a large number of large images that the memory savings is significant. It would be appropriate to use "framing_mode==5" for storage of things like tomographic slices.

When "framing_mode==6", then "draw mode" decoders can treat the frame in the same way as when "framing_mode==3". If "LOCA" chunks appear, they must appear before the images to which they per-

tain are made visible. A "sprite mode" decoders will have to revert to "draw mode" to display this frame type, or or else it will have to make its own clones and reassign them to unique, properly ordered unused levels. Encoders are encouraged to use "framing_mode==3" instead, unless the frame contains such a large number of large images that the memory savings is significant.

The "frame_duration" field gives the duration of display, which is the minimum time that must elapse from the beginning of displaying one frame until the beginning of displaying the next. It is measured in "ticks" using the tick length determined from "ticks_per_second" defined in the "MHDR" chunk.

This overrides the value of frame_duration given in the "MHDR" chunk. The value of "frame_duration" will remain in effect until another "FRAM" chunk redefines it or until a "SEEK" chunk is encountered, when the duration reverts to the value from "MHDR" or "SAVE".

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

A viewer does not actually have to follow the procedure of erasing the screen, redisplaying the background, and recompositing the images against it, but what is displayed when the frame is complete must be the same as if it had. It is sufficient to redraw the parts of the display that change from one frame to the next.

The "sync_id" list provides a point at which the processor must wait for all pending processes to reach the synchronization point having the same "sync_id" 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. When the period defined by the sum of the "frame_duration" and the "sync_timeout" fields elapses, processing can resume even though the processor has not received an indication that other processes have reached the synchronization point.

Note that the synchronization point does not occur immediately, but at the end of the frame that follows the "FRAM" chunk. If it is necessary to establish a synchronization point immediately, this can be done by using two consecutive "FRAM" chunks, the first setting a temporary "frame_duration==0", "sync_timeout", and "sync_id", and the second establishing the synchronization point:

```
FRAM 0 1 1 1 0000 sync_timeout sync_id
FRAM
```

The identifier "sync_id==0" is reserved to represent synchronization with a user input from a keyboard or pointing device. If multiple channels (not defined in this specification) are not present, viewers can ignore other values appearing in the "sync_id" list.

2.3.5 SHOW Show images

The "SHOW" chunk can be used to show or hide one or more previously-defined images, without making any changes except possibly for the location. It contains a visibility byte and two two-byte "image_id" fields:

```
1 byte:  visibility (unsigned integer)
     0:  make the images visible
     1:  make the images invisible
2 bytes: first_id (unsigned integer).
2 bytes: last_id  (unsigned integer).
```

When "visibility==0", the images are shown in the order "first_id" through " last_id". If "first_id > last_id" then the images are shown in reverse order (this is not permitted when "framing_mode==2" or "framing_mode==3" in the "FRAM" chunk). When "framing_mode==1" or when the "FRAM" chunk is not present, each image is displayed as a separate frame. When "visibility==1", the images are made invisible (this is not permitted when "framing_mode==2" or "framing_mode==3" in the "FRAM" chunk).

An instance of each image will be displayed at the location specified by the "LOCA" chunk, or at {0,0} if no "LOCA" chunk is present with the same "image_id". When the "LOCA" chunk is used in the delta form, which will frequently be the case, each image must be displaced from its previous position by the values given in the "LOCA" chunk.

It is not necessary to follow an "IHDR–IEND" or "DHDR–DEND" sequence with a "SHOW" chunk. The "SHOW 0" chunk is implied, and such images are always displayed if they are within clipping boundaries of the frame, unless the sequence was preceded by the "DEFN" chunk.

2.4 Ancillary MNG chunks

This section describes ancillary MNG chunks. MNG-compliant decoders are not required to recognize and process them.

2.4.1 sPLT Suggested palette

[This can be removed from the MNG specification once the sPLT chunk (the proposed spAL chunk) is registered as a PNG chunk. What follows is a copy of the present proposal, which describes the chunk in its unregistered form, using the name "spAL" instead of "sPLT"]

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

This chunk's contents are a zero-byte-terminated text string that names the palette [followed by a 20-byte signature and a zero-byte terminator] and a 1-byte spAL_sample_depth integer, followed by a series of palette entries, each a six-byte or ten-byte series, containing five unsigned integers:

```
n bytes:     (Latin-1 text), palette_name.
1 byte:      (null) terminator.
```

```
[ 20 bytes:  signature ("PNG group 1996-10-22").
  1 byte:    (null) separator. ]

1 byte:      (unsigned integer), spAL_sample_depth.
             Must be 8 or 16.

1 or 2 bytes: (unsigned integer), red intensity.
           0:  black
           etc.
           255 or 65535: full red intensity

1 or 2 bytes: (unsigned integer), green intensity.

1 or 2 bytes: (unsigned integer), blue intensity.

1 or 2 bytes: (unsigned integer), alpha.
           0: fully transparent
           etc.
           255 or 65535: fully opaque

2 bytes:      (unsigned integer), frequency.
              (relative frequency of occurrence).
etc.
```

There can be any number of entries; a decoder determines the number of entries from the remaining chunk length after the "palette_name" field and its zero-byte terminator [, the "signature" and its zero-byte terminator], and the `spAL_sample_depth` byte. This length not divisible by six (if `spAL_sample_depth==8`) or by ten (if `spAL_sample_depth==16`) is an error. Entries must appear in decreasing order of "frequency".

The "palette_name" (e.g. "256 color including Macintosh default", "256 color including Windows-3.1 default", "Optimal 512") identifies the palette, which can permit applications or people to choose the appropriate one when more than one suggested palette appears in a PNG file. The "palette_name" string must follow the format of a `tEXt` keyword: It must consist only of printable Latin-1 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.

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

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

The palette uses 8 bits or 16 bits (1 or 2 bytes) per value regardless of the image bit depth specification, according to the number given in the spAL_sample_depth field. Decoders wishing to construct a palette with a different bit depth can accomplish this by scaling the RGBA entries, as described under "[link] sample depth rescaling" in the PNG specification. The palette samples have the same gamma and chromaticity values as those of the pixel samples.

Note: Earlier versions of the PNG specification recommended that the PLTE [link to PLTE] and hIST chunks be used for this purpose. While this is still allowed, to maintain backward compatibility, the spAL chunk is preferable, particularly when transparent pixels are present. When both the PLTE and spAL chunks are present, the PLTE data should only be used for decoding the indexed-color (color type 3) pixels, and the spAL data should be used for constructing the display palette, if the viewer is not capable of displaying the number of colors present in the PLTE chunk. If the hIST chunk is also present, decoders that process the spAL chunk should ignore the hIST chunk.

[from here on the writeup differs from that in the PNG "sPLT" proposal]

This chunk can appear for any color type. There can be multiple "spAL" chunks in a MNG datastream. If a "palette_name" is repeated, the previous palette having the same "palette_name" is replaced. It is not permitted, at the MNG top level, to redefine a palette after the "SAVE" chunk with the same "palette_name" as one that appears ahead of the "SAVE" chunk.

When a decoder needs to chose between a suggested palette defined at the MNG level and a suggested palette defined in the PNG datastream (either with the "spAL" chunk, or with the "PLTE/hIST" chunks for grayscale or truecolor images), it should give preference to the palette from the MNG level, to avoid spurious frame-to-frame color changes.

MNG editors that write PNG files should ignore the "spAL" data from the MNG level and simply copy any "spAL" chunks appearing within the PNG datastreams.

When a "SEEK" chunk is encountered, all "spAL" data will be restored to the state that existed when the "SAVE" chunk was encountered.

2.4.2 fPRi Frame priority

The "fPRi" chunk allows authors to assign a priority to a portion of the MNG datastream. Decoders can decide whether or not to decode and process that part of the datastream based on its "priority" compared to some measure of "cost."

The "fPRi" chunk contains three bytes:

```
1 byte: fPRi_delta_type (unsigned integer).
     0: Values are given directly
     1: Values are determined by adding
        the fPRi data to the previous
        values, modulo 256
```

```
1 byte: priority (unsigned integer). Value to
        be assigned to subsequent frames
        until another vALu chunk is reached

1 byte: max_priority (unsigned integer).  Maximum
        priority that will appear in any subsequent
        fPRI chunk.
```

While 256 distinct values of "priority" are possible, it is recommended that only the values 0 (low priority), 128 (medium priority), and 255 (high priority) be used. Viewers that can only display a single image can look for one with "priority==255" and stop after displaying it. If the datastream contains a large number of frames and includes periodic "initial" frames that do not contain PND datastreams, the "initial" frames could be preceded by a "fPRi" with "priority==128" and followed by one with "priority==0", and the best representative frame could be preceded by a "fPRi" chunk with "priority==255". Then single-image viewers would just display the representative frame, slow viewers would display just the "initial" frames, and fast viewers could display everything.

It is not permissible for a portion of the datastream to depend on any portion of the datastream having a lower value, because a decoder might have skipped the lower value portion. Use of the "fPRi" chunk is illustrated in Examples 1 and 4.

When a "SEEK" chunk is encountered, the value of "priority" reverts to its value when the "SAVE" chunk was processed. Viewers that care about the priority can assume "priority==255" for any portion of the MNG datastream that is processed prior to the first "fPRi" chunk.

2.4.3 PNG ancillary chunks

The namespace for MNG chunk names is separate from that of PNG. Only those PNG chunks named in this section are also defined the MNG top level. They have exactly the same syntax and semantics as when they appear in a PNG datastream:

- "tEXt"
- "tIME"
- "zTXt"

The following PNG chunks are also defined at the MNG top level. They provide default values to be used in case they are not provided in a subsequent PNG datastream:

- "cHRM"
- "gAMA"
- "sRGB" (assuming sRGB gets registered)

The following PNG chunk is also defined at the MNG top level. It provides values that take precedence over those that might be provided in a subsequent PNG datastream:

- "sPLT" [same as PNG proposed sPLT (or spAL) chunk] (this chunk also takes precedence over the "PLTE" chunk in a subsequent PNG datastream when the "PLTE" and "hIST" chunks chunk is being used as a suggested palette (i.e. "color_type != 3").

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 in this specification 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 should use a different file signature and filename extension, or they can wrap it in a MNG datastream consisting of the MNG signature, the "MHDR" chunk, a "BLNK" chunk with the appropriate dimensions, the PND datastream, and the "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, another PND datastream, or it could have been generated by a "BLNK" chunk or a PNG-like datastream introduced by a "BASI" chunk.

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

This section describes critical PND chunks. MNG-compliant decoders must recognize and process them.

3.1.1 DHDR PND datastream header

The "DHDR" chunk introduces a PND datastream. Subsequent chunks, through the next "DEND" chunk, are interpreted according to the PND format.

The "DHDR" chunk can contain 4, 12, or 20 bytes:

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

1  byte: image_type.

       0: Image type is unspecified.  An IHDR or IPNG
          chunk must be present.

       1: Image type is PNG.  IHDR and IPNG can be
          omitted under certain conditions.

1  byte: delta_type.

       0: entire image replacement

       1: block pixel addition, by samples, modulo
          2^bit_depth.

       2: block 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: block pixel replacement

       4: block alpha replacement

       5: no change to pixel data

4 bytes: block_width (unsigned integer).  Omit when
         delta_type==5.

4 bytes: block_height (unsigned integer).  Omit when
         delta_type==5.

4 bytes: block_x_location (unsigned integer),
         measured in pixels from the left edge of
         the basis image.  Omit when delta_type==0
         or when delta_type==5.
```

```
4 bytes: block_y_location (unsigned integer),
         measured in pixels from the top edge of
         the basis image.  Omit when delta_type==0
         or when delta_type==5.
```

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

When "delta_type==0", the width and height of the generated image are given by the block_width and block_height fields.

For all other values of "delta_type", the width and height of the generated image are inherited from the basis image.

When "delta_type==1, 2, 3, or 4", 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.

Entire image replacement When "delta_type==0" in the "DHDR" chunk, the pixel data in the "IDAT" chunks represent a completely new image, with dimensions given by the "block_width" and "block_height" fields of the "DHDR" chunk. Data from chunks other than "IDAT" can be inherited from the basis image.

Block pixel addition When "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 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 as those of the basis image.

[We might want to add a compression_type that uses deflate with the final 32 kilobytes of 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. Only the pixels in the block defined by the block location and dimensions given in the "DHDR" chunk are changed. The size of the "IDAT" data must correspond exactly to this rectangle.

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==3", the deltas are differences between index values, not between color samples.

Block alpha addition 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 "bit_depth" of the data must match that of the basis image, and "color type" is 0 (grayscale), regardless of the "color_type" of the basis image. 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.

Block pixel replacement When the "delta_type==3" in the "DHDR" chunk, the pixel data in the "IDAT" chunks represent replacement values for the pixel samples in the rectangle given by the block location and dimension fields in the "DHDR" chunk.

The "bit_depth" and "color_type" of the data must match that of the basis image. The compression method, filter method, and interlace method need not be the same.

Block alpha replacement When the "delta_type==4" in the "DHDR" chunk, the pixel data in the "IDAT" chunks represent replacement values of the alpha samples in the rectangle given by the block location and dimension fields in the "DHDR" chunk. The "bit_depth" of the data must match that of the basis image, and "color type" is 0 (grayscale), regardless of the "color_type" of the basis image. 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 "delta_type==5" in the "DHDR" chunk, there is no change to the pixel data, and it is an error for "IDAT" to appear.

3.1.2 DEND End of PND datastream

End of PND datastream. A "DEND" chunk must be present for each "DHDR" chunk in a MNG datastream.

The "DEND" chunk is empty.

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/IPNG, PNG chunks, IEND

An incomplete PNG (Portable Network Graphics) datastream. The basis image must be a PNG or PNG-based PND. The datastream can be introduced by a complete PNG "IHDR" chunk or by an "IPNG" chunk, which is empty.

If the "IHDR" chunk is present, its "width", "height", "bit_depth", and "color_type" fields are ignored. The values for these parameters are inherited from the basis image or from the "PROM" chunk.

The "compression_method", "interlace_type", and "filter_type" fields, if different from those of the basis image, are used in decoding any subsequent "IDAT" chunks, and the new values will be inherited by any subsequent image that uses this image as its basis.

See the PNG specification for the format of the PNG chunks. The PNG datastream must contain at least "IHDR" and "IEND" (whether actually present in the datastream or omitted and included by implication, as described below) 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.

The "IPNG" chunk can be used instead of the "IHDR" chunk if the "IHDR" chunk is not needed for resetting the value of "compression_method", "filter_type", or "interlace_type". The purpose of this chunk is to identify the beginning of the PNG datastream, so decoders can start interpreting PNG chunks instead of PND chunks. The decoder must treat this datastream as though the "IHDR" chunk were present in the location occupied by the "IPNG" chunk.

The "IHDR" chunk can also be omitted when "image_type==1" and the PNG stream begins with either a "PLTE" chunk or an "IDAT" chunk. In this case, no "IPNG" chunk is required, either. The decoder must

treat this datastream as though the "IHDR" chunk were present, immediately preceding the first PNG chunk. If the first PNG chunk is neither a "PLTE" chunk nor an "IDAT" chunk, then either the "IPNG" or "IHDR" must be present to introduce the PNG datastream.

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 are the raw pixel data that existed prior to any transformations that were applied while displaying the previous frame.

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

MNG viewers are expected to ignore the "pHYs" and "oFFs" chunks. MNG editors are expected to recognize and copy the "pHYs" chunk, because it is a known ancillary chunk that appears in the PNG specification, and to treat the "oFFs" chunk as an unknown chunk that will be handled as described in Paragraph 3.1.5, below.

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 "INHR" chunk is not present, known chunks (always including all standard chunks described in the PNG specification) are considered to have appeared in their proper order with respect to the critical chunks. Unknown chunks are ordered as described in Paragraph 3.1.5, below. When the "INHR" chunk is present, then all chunks are considered to have appeared in the order given by the "INHR" chunk.

The "IEND" chunk can be omitted, if it would be the last chunk in the PND datastream before the "DEND", or when no PNG chunks are present.

3.1.5 INHR Inherited chunks

The "INHR" chunk provides rules governing how chunks are to be inherited from the basis image and provides a template for ordering the chunks in an output PNG datastream.

Viewers that only process public PNG chunks and are not concerned about selecting chunks such as "tEXt" or "zTXt" by keyword also need not process the "INHR" chunks, although they must recognize them. Applications that write PND streams containing only public PNG chunks need not be concerned about writing "INHR" chunks.

The "INHR" chunk contains a series of five-byte entries, each giving a chunk name and a rule.

```
4 bytes: chunkname

1  byte:  rule.

      0: (Replace) Inherit all instances of the chunk
         in the basis image.  If a new instance is found
         in the PND datastream, delete all inherited
         instances and replace them with the new instance
         or instances.
```

```
1: (Append) Inherit all instances of the chunk in
   the basis image.  Append any new instances found
   in the PND datastream.

2: (Ignore) Ignore any instances of the chunk in
   the basis image.  Insert any new instances found
   in the PND datastream.

3: (Replace by Keyword) The chunk is one for which
   multiple instances are permitted, and the first
   field of the chunk is a null-byte terminated
   Latin-1 keyword or name (tEXt, zTXt, spAL, for
   example).  Inherit all instances of the chunk
   from the basis image.  If a new instance is
   found in the PND datastream, delete all
   inherited instances having the same keyword or
   name and replace them with the new instance or
   instances.

4: (Special) Inheritance and placement of this
   chunk is governed by special rules that must be
   known to the PND decoder (IDAT, for example)
```

The "INHR" chunk provides a list of chunknames that are to be inherited or not inherited from the basis image, regardless of the copy-safe rules, and regardless of whether the chunk is recognized or not. The number of entries is determined from the chunk length, divided by 5.

The "INHR" chunk also gives a template for the arrangement 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 unknown but copy-safe chunks, it should use the "INHR" chunk data as a template to ensure that they are properly placed.

The "INHR" chunk can be omitted. If no "INHR" chunk appears in a PND datastream or is associated with its basis image, then data from the global "INHR" chunk, defined Paragraph 2.2.8, above, can be used if it is present. PND applications are expected to know the copying rules and order of placement of those chunks defined in the core specification for the image format corresponding to the "DHDR" "image_type", regardless of whether an "INHR" chunk is present and whether those chunks are listed in it. In particular, the PNG "IHDR" and "IEND" chunks need not be listed in the "INHR" chunk, because the position of these chunks is already well-defined. "PLTE" (with "rule==0") and "IDAT" (with "rule==4") should be listed, however, because other chunks need to be located with respect to them. It is not an error for a chunkname to appear in the list, when that chunk does not appear in the basis image or in the PND datastream.

When an unknown chunk is found in the PND datastream that appears in the basis image but is not listed in the "INHR" chunk, the decoder can copy the chunk into the same position relative to critical chunks that it occupied in the basis image. When an unknown chunk appears neither in the basis image nor in the "INHR" list, or when a chunk's "rule==4" and the PND decoder does not know the special copying rules for the chunk in question, the decoder must treat it as an unknown chunk, ignoring it if it is ancillary, and abandoning the PND datastream if it is critical.

The "INHR" chunk data itself is inherited, so an "INHR" chunk need only appear in the first of a sequence of PND datastreams that have the same "image_id", if there are no changes. When a "INHR" chunk is inherited and also appears on the PND datastream, the entire list of chunk names and rules is replaced.

Applications that reconstruct a PNG datastream from a PND datastream and a basis PNG or PND datastream must not write the "INHR" chunk itself to the resulting PNG datastream, because the "INHR" chunk would not be recognized by a PNG decoder.

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

3.1.6 SBYK Select chunks by keyword

The "SBYK" chunk selects chunks for inheritance from the basis image by the keyword.

Viewers that only process public PNG chunks and are not concerned about selecting chunks such as "tEXt / zTXt" by keyword need not process "SBYK" chunks, although they must recognize them. Applications that write PND streams containing only public PNG chunks need not be concerned about writing "SBYK" chunks.

The chunk is either empty or it contains a select/reject byte, a chunk name, and a keyword.

```
1 byte:   sbyk_select (unsigned integer)
     0:   select chunks by keyword
     1:   reject chunks by keyword
4 bytes:  (four ASCII bytes), chunk name.
n bytes:  (Latin-1 string), keyword.
```

When the "SBYK" is empty, the "SBYK" list is cleared.

When the "SBYK" chunk is not empty, the specified {sbyk_select, chunk name, keyword} triple is added to the list associated with the "image_id" that is being modified.

The chunk must be one whose first field is Latin-1 string and whose second field is a null separator byte, such as the "tEXt" chunk, which begins with an Latin-1 keyword, or the proposed "fALS" chunk, which begins with an Latin-1 "purpose" string.

The keyword need not be terminated by a null byte; if it is, the null byte will be ignored. The keyword must follow the format of a "tEXt" keyword: It must consist only of printable Latin-1 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. Keywords are case-sensitive. Use caution when printing or displaying keywords (Refer to Security considerations, Chapter 8, below).

Any chunks in the basis image having the same chunk name and keyword are selected (if "sbyk_select==0") or rejected (if "sbyk_select==1") for inclusion in the resulting PND image. It is not an error to list a {chunkname, keyword} pair that does not exist in the basis image.

The "SBYK" chunk can be omitted. If no "SBYK" chunk appears in a PND datastream or is associated with its basis image, then data from the global "SBYK" chunk, defined Paragraph 2.2.9, above, can be used if it is present.

The keyword, "purpose," or other identifying string must appear *first* in the chunk's data segment. The practice of putting a version identifying "signature" first in the data segment of unregistered experimental chunks will defeat this mechanism, so it is better to place such a "signature" after the keyword and its zero-byte separator, if you wish to use the "SBYK" mechanism with your experimental chunk.

Applications that reconstruct a PNG datastream from a PND datastream and a basis PNG or PND datastream must not write the "SBYK" chunk itself to the resulting PNG datastream, because the "SBYK" chunk would not be recognized by a PNG decoder.

Multiple instances of the "SBYK" chunk are permitted in a PND datastream. Any instances must appear prior to the "IHDR" chunk, if the "IHDR" chunk is present.

3.2 PND ancillary chunks

This section describes ancillary PND chunks. MNG-compliant decoders are not required to recognize and process them.

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==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==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==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==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 PNG chunks

The following PNG chunks are also defined at the PND level, and have exactly the same syntax and semantics as those in a PNG stream:

- "tEXt"
- "tIME"
- "zTXt"

The PNG "oFFs" and "pHYs" chunks must be ignored by MNG viewers and simply copied by MNG editors.

4 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" chunk or for possible use as the basis image for a subsequent PND datastream.

The following information must be retained, for each "image_id" that is defined and not subsequently discarded:

- The data from the "LOCA" chunk.
- The pixel data and the values associated with other recognized PNG chunks such as "PLTE" and "gAMA", subject to the chunk copying rules and the "INHR" chunk.
- The contents of the "INHR" chunk and "SBYK" chunks associated with the "image_id", if the decoder needs this information.

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" or the "SEEK" chunk.

5 Decoder handling of fatal errors

When a fatal error is encountered, such as an unknown critical MNG chunk, MNG viewers should attempt to recover gracefully by abandoning processing of the frame and searching for a "SEEK" chunk. If errors occur before the "SAVE" chunk is reached, the viewer should probably just abandon the MNG datastream.

When an error occurs within a image datastream, such as an unknown critical PNG chunk or a missing basis image where one was required, only that image should be abandoned and the associated "image_id" should be discarded.

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 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.

7 Decoder handling of palettes

When a "PLTE" chunk is received, it only affects the display of the PNG or PND datastream that includes it, and any subsequent PND datastreams that depend on it.

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 resulting image is displayed using the old pixel samples as indices into the new palette, which provides a "palette animation" capability.

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" "image_id" field to depend on the other.

A composite frame 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 "spAL" 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 or a misplaced "sync_id" with a long "sync_timeout" value. 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 "sPLT" 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 "sPLT" "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

We use the "#" character to denote commentary in these examples; such comments are not present in actual MNG datastreams.

9.1 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
     1010880   # max storage is 720x468x3
     8         # four unused bytes, max_bit_depth 8
     3         # Max 3 samples per pixel (color type 2)
     0 0       # Not OK discard, max_image_id
     000000..  # Fifteen reserved bytes
tEXtTitle\0Sample Movie
fPRI 128 255   # default frame priority is "medium"

SAVE
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   # A PNG-delta frame
IDAT ...       # The IDAT gives the 20x30 block
DEND           # of deltas
```

```
DHDR 0 1 1 20 30 102 222     # 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
fPRI 255 255    # This is the representative frame that
IHDR 720 468 ...# will be displayed by single-frame
IDAT ...        # viewers.
IEND
fPRI 128 128   # Return to medium frame priority

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

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

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: In full-sprite-model, this has to be redone to use 3 different image_ids

```
\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
     0000     # Max storage is undefined
     16 1     # Max depth 16, max 1 sample per pixel
     0 2      # Not ok to discard, max_image_id
     000000.. # Fifteen reserved bytes
BACK 0 0 64 64 192   # Sky blue background
FRAM 1 0 0 0         # Composite frame, 1024 x 512
LOCA 0 0 6 6     # Location of first (bottom) image
DEFI 0
IHDR 500 500 16 0 .. # A 16-bit graylevel image
gAMA 50000
IDAT ...
IEND                 # End of image
```

```
CLON 0 1 0           # Make a full clone
LOCA 1 0 518 6    # Location to display the clone.
DHDR 1 1 3 0 0 0 0   #  Modify the clone.
INHR gAMA 0 tEXt 3 faLT 0 IDAT 4 # establish chunk order
gAMA 100000   # gamma value is 100000 (gamma=1.0)
tEXtComment\0The faLT chunk is described in ftp://swrinde....
faLT ...             # Apply pseudocolor to basis image
DEND                 # End of image
SHOW 0 1 1
LOCA 2 0 900 400 # Overlay near lower right-hand corner
DEFI 2
IHDR 101 101 2 3 ...
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

MEND                 # End of MNG datastream
```

9.3 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
     600000       # Max storage is 2x512x512x1 + 32x32x1
     8            # max_bit_depth 8
     1            # Max samples per pixel
     0 3          # Not ok to discard, max_image_id
     000000..     # Fifteen reserved bytes
FRAM 1 0 0 0      # First frame
DEFI 1            # Define image 1  (LOCA 1 0 0 0 implied)
IHDR 512 512 ... # It's a full-display PNG image
etc               # Chunks according to PNG spec
IEND              # SHOW 0 1 1 is implied by DEFI 1
DEFI 2            # Define image 2
LOCA 2 0 300 200    # Location for image 2
IHDR 32 32 ...    # It's a small PNG
gAMA 50000
IDAT ...
DEND              # IEND is omitted
FRAM              # Start new frame
                  # New location for image 1 is still 0,0
SHOW 0 1 1        # Display image 1 from previous frame
LOCA 2 1 10 5     # New (delta) location for image 2
SHOW 0 2 2          # Retrieve image 2 from previous frame,
CLON 2 3 0        # make a full clone of it as image 3
LOCA 3 0 400 500 # Location for image 3
DHDR 3 1 3 0 0 0 0      # Modify image 3
tRNS ...          # Make it semitransparent
DEND              # SHOW 0 3 3 is implied by DHDR
FRAM              # Next frame (repeat this FRAM-SHOW 3 3
                  #   sequence with different locations to
                  #   move the images around)
                  # New location for image 1 is still 0,0
SHOW 0 1 1        # Display image 1 from previous frame
LOCA 2 1 10 5     # New (delta) location for image 2
SHOW 0 2 2          # Show image 2 from previous frame,
LOCA 3 1 5 -2     # New location for image 3
SHOW 0 3 3          # Show image 3
FRAM              #  Another frame
etc.
FRAM
etc.              # More frames
MEND              # End of MNG datastream
```

9.4 Example 4: "Fading in" a transparent image

The opaque parts of this image will "fade in" gradually. This technique will not 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. This example also illustrates the use of the "fPRi" 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
     10000     # 64x64x1 x 2 is 8192
     8 1       # Max bit depth 8, 1 sample per pixel
     0 2      # Not ok to discard, max_image_id
     000000.. # Fifteen reserved bytes
BACK 1 0 192 192 192 # "Browser gray" default background
DEFN 1      # No FRAM chunk means framing mode 0
IHDR ...    # PNG header
PLTE ...
tRNS ...    # Entries are zero for the transparent color
            # and 255 for the nontransparent ones.
IDAT ...
IEND
fPRi 0 0 255  # Give the fade-in sequence a low priority
CLON 1 2 0  # Make a working copy of the image because
            # it will be modified during the low-priority
            # part of the datastream.  It's a full clone.
DHDR 2 1 3 64 64 0 0
fADE 0 254  # Subtract 254 to alpha for all nontransparent
            # pixels to bring them down from 255 to 1
LOOP 0 0 15
DHDR 2 1 3 64 64 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=241
            # and the transparent ones will still be 0.
DISC 2      # Discard the working copy
fPRi 0 255 255  # Give the final frame the highest value
FRAM 0 1 0 0 60 # Hold the last frame for at least 60 ticks
            # (2 sec).  Applications might show it longer.
SHOW 0 1 1  # This copy still has alpha=255 for the
            # opaque pixels and alpha=0 for the others.
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 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)
     50000       # Max storage is 150 x 150 x 2 = 45000
     16 1        # Max bit depth 8, 1 sample per pixel
     0 0      # Not ok to discard, max_image_id
     000000.. # Fifteen reserved bytes

tEXtTitle\0Weather modeling results
tEXtComment\0The pcAL, xsCL, ysCL, zsCA, and tsCL chunks
 in this file are written according to the PNG Sci-vis
 chunks specification version 19960921 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
zsCAHeight (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 4 0 0 0     # 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 # Block is entire image
IDAT ...     # IHDR is omitted; everything matches top
DEND         # IEND is also omitted
etc.         # Repeat DHDR through DEND 148 more times
SEEK 3588720 4621885
FRAM         # End of first block
etc.         # Repeat FRAM through SEEK 19 more times
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.

"Draw" mode version:

```
\211 M N G \r \n ^z \n  # MNG signature
MHDR 1024 768 128 64 .. 2 .. # Start of MNG datastream
FRAM 4 0 0 0
LOCA 1 0 0 -64 # Set up an offscreen copy of the tile
DEFN 1         # Give it ID==1, don't show it immediately
IHDR 128 64 ...
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 1 0 64 # Move the first copy down 64 rows
SHOW 0 1 1     # Display it
CLON 1 2 1     # Create a partial clone of the tile
LOOP 1 0 7     # X loop - 7 additional columns
LOCA 2 1 0 128 # Move it to the right 128 columns
SHOW 0 2 2     # Use the second copy
ENDL 0
ENDL 1
MEND
```

To make a "sprite" mode version: We must unroll the loops and make 95 partial clones:

```
\211 M N G \r \n ^z \n  # MNG signature
MHDR 1024 768 128 64 .. 96 .. # Start of MNG datastream
FRAM 3 0 0 0
LOCA 1 0 0 0  # Set up the corner tile
DEFN 1        # Give it ID==1, don't show it immediately
IHDR 128 64 ...
PLTE ...
IDAT ...
IEND ...
LOCA 2 1 0 128 # Move clone 2 to the right 128 columns
CLON 1 2 1     # Make partial clone with image_id 2
LOCA 3 1 0 128 # Move clone 3 to the right 128 columns
CLON 2 3 1     # Make partial clone with image_id 3
etc.
LOCA 13 1 0 64 # Move clone 13 down 64 rows
CLON 1 13 1    # Make partial clone with image_id 13
etc.
SHOW 0 1 96      # Show the image and its partial clones
MEND
```

9.7 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.
     800000      # Max storage 512x3000x(1/8) + 512x256x4
     4 1         # Max bit depth 4, 1 sample per pixel
     0 2      # Not ok to discard, max_image_id
     000000.. # Fifteen reserved bytes
FRAM 1 0 0 0
DEFN 1            # Define image 1 but don't display now
LOCA 1 0 0 256    # Initially it's offscreen, just
                  # below the 512 by 256 window
IHDR 512 3000 1 0 ... # A PNG datastream containing the
PLTE ...          # text (or whatever) to be scrolled
IDAT ...
IEND

DEFI 2
LOCA 2 0 0 0     # This could be omitted
IHDR 512 256 8 6 ... # A PNG datastream containing some kind
PLTE ...         # of alpha-blended border that is
tRNS ...         # transparent in the center
IDAT ...
IEND

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

MEND
```

9.8 Example 8: Converting a GIF animation to MNG

Outline of a program to convert GIF animations to MNG format:

```
begin
    write "MHDR" chunk with max_image_id=65535
    write "BACK" chunk
    saved_images := 0
    frame_duration := 0
    first_frame := TRUE
    for subimage in gif89a file do
       if(frame_duration != gif_duration) then
         frame_duration := gif_duration
         write "FRAM 1 2 0 0 frame_duration" chunk
         first_frame := FALSE
       else if(first_frame == TRUE)then
         write "FRAM 1 0 0 0" chunk
         first_frame := FALSE
       else
         write "FRAM" chunk
       endif

       write "DEFN saved_images" chunk
       write "<image>
       write "SHOW 0 0 saved_images" chunk

       if (gif_disposal_method == 0
          OR gif_disposal_method == 2) then
          /* (undefined or restore background) */
             write "DISC" chunk
             saved_images := 0
       else if (gif_disposal_method == 1) then
          /* (keep) */
          saved_images := saved_images + 1
       else if (gif_disposal_method == 3) then
          /* (restore previous) */
          write "DISC saved_images" chunk
       endif

    endfor
    write "FRAM" chunk
    write "MEND" chunk
end
```

Where "<image>" represents a PNG or PND datastream containing the GIF frame converted to PNG format.

Caution: if you write such a program you might have to pay royalties in order to use it or convey it to anyone else.

10 Credits

Contributors' names are presented in alphabetical order:

- Mark Adler, madler@alumni.caltech.edu
- Thomas Boutell, boutell@boutell.com
- John Bowler, jbowler@acm.org
- 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:

- GIF is a service mark of CompuServe Incorporated.
- 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 01 May 1997