PNG-GROUP-DRAFT
MNG
Expires: 04 Jul 1997

PNG Development Group
mpng-list@dworkin.wustl.edu
04 Jan 1997

MNG (Multiple-image Network Graphics) Format Version 19970104

Thirty-first draft, file: draft-mng-19970104.txt

Status of this Memo

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

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

Comments on this document can be sent to the PNG specification maintainers at one of the following addresses:

- mpng-list@dworkin.wustl.edu
- png-group@w3c.org
- png-info@uunet.uu.net

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 thirtieth MNG draft (draft-mng-19961231)

- Allow BASI image to be "abstract."
- Removed 90 and 270 degree rotations from PAST chunk.
- Changed "modifiability" to "abstract_flag"
- Made "abstract" the default (zero) image type
- Changed "target_id" to "destination_id" in PAST chunk description.
- Added "target_x" and "target_y" fields to PAST.

- Added "fill_method" byte to PROM.
- Added information about "abstract" and "concrete" images to the introduction.
- Rewrote parts of the FRAM chunk description for clarity. It still needs more work.
- Changed SAVE chunk to the FREZ chunk, and removed requirement for SEEK chunk to restore information.
- Added eXPI chunk (stolen from DOH) and deleted tHUM.
- Added some very simple examples.

Abstract

This document presents the *[proposed]* format of a MNG (Multiple-image Network Graphics) datastream. MNG is a multiple-image member of the PNG (Portable Network Graphics) format family, 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 examples that demonstrate various capabilities of MNG including simple movies, composite frames, loops, fades, tiling, scrolling, storage of voxel data, converting GIF animations to MNG format.

Contents

1 Introduction 4
2 MNG chunks 7
2.1 Critical MNG control chunks 7
2.1.1 `MHDR` MNG datastream header 7
2.1.2 `MEND` End of MNG datastream 7

2.1.3 FREZ Freeze information . . . 8
2.1.4 SEEK Seek point . . . 8
2.1.5 LOOP, ENDL Define a loop . . . 9
2.2 Critical MNG image defining chunks . . . 10
2.2.1 DEFI Define an image . . . 10
2.2.2 IHDR, PNG chunks, IEND . . . 12
2.2.3 BASI, PNG chunks, IEND . . . 12
2.2.4 DHDR, PND chunks, DEND . . . 14
2.2.5 CLON Clone an image . . . 14
2.2.6 PAST Paste an image into an another . . . 16
2.2.7 DISC Discard images . . . 17
2.3 Critical MNG image displaying chunks . . . 18
2.3.1 LOCA Image location . . . 18
2.3.2 CLIP Image clipping boundaries . . . 19
2.3.3 FRAM Delimit a frame . . . 20
2.3.4 SHOW Show images . . . 25
2.4 Ancillary MNG chunks . . . 26
2.4.1 bACk Background . . . 26
2.4.2 eXPI Export image . . . 27
2.4.3 fPRi Frame priority . . . 27
2.4.4 nEED Resources needed . . . 28
2.4.5 PNG ancillary chunks . . . 29
3 The PND format . . . 30
3.1 PND critical chunks . . . 31
3.1.1 DHDR PND datastream header . . . 31
Entire image replacement . . . 32
Block pixel addition . . . 32
Block alpha addition . . . 33
Block pixel replacement . . . 33
Block alpha replacement . . . 33
No change to pixel data . . . 34
3.1.2 DEND End of PND datastream . . . 34
3.1.3 PROM Promotion of basis image . . . 34
3.1.4 IHDR/IPNG, PNG chunks, IEND . . . 34
3.1.5 INHR Inherited chunks . . . 36
3.1.6 SBYK Select chunks by keyword . . . 37
3.2 PND ancillary chunks . . . 38
3.2.1 fADE Fade in or out . . . 38
4 Recommendations for encoders . . . 39
5 Recommendations for decoders . . . 39
5.1 Retaining image data . . . 39
5.2 Decoder handling of fatal errors . . . 40
5.3 Decoder handling of interlaced images . . . 40

5.4 Decoder handling of palettes 40
6 Security considerations 40
7 Appendix: Examples 42
7.1 Example 1: A single image 42
7.2 Example 2: Very simple movie 42
7.3 Example 3: Simple slideshow 43
7.4 Example 4: A more efficient slideshow 43
7.5 Example 5: Simple movie 44
7.6 Example 6: Single composite frame 45
7.7 Example 7: Movie with sprites 46
7.8 Example 8: "Fading in" a transparent image 47
7.9 Example 9: Storing three-dimensional images 48
7.10 Example 10: Tiling 48
7.11 Example 11: Scrolling 50
7.12 Example 12: Converting a GIF animation to MNG 51
8 Credits 53

1 Introduction

This *[proposed]* specification defines the format of the MNG (Multiple-image 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://www.wco.com/~png/`

A MNG datastream describes a sequence of single frames, each of which can be composed of one or more images.

An image is defined by

- A PNG datastream as defined in the PNG specification.
- A PND datastream (A PNG-Delta, as defined herein).
- A MNG "`BASI`" chunk.
- A MNG "`PAST`" chunk.
- A reference in a "`SHOW`" chunk to a previously-defined image.

A MNG datastream consists of

- The 8-byte MNG signature.
- A 12-byte "`MHDR`" chunk.

- One or more frame definitions. A frame can be
  - A single image.
  - A group of images delimited by "FRAM" chunks.
- "LOOP-ENDL" chunks that enclose frame definitions.
- "SEEK" chunks that mark points in the datastream where processing can be restarted.
- Various chunks for creating and manipulating images.
- The "MEND" chunk.

Images can be "concrete" or "abstract". The distinction allows decoders to use more efficient manipulation of images when it is not necessary to retain the image data in its original form or equivalent in order to show it properly on the target display system.

- A *concrete* image must be stored by the decoder in a form that retains the complete image description, sufficient to regenerate the original image description or its equivalent without loss. In the case of a PNG image, this means that the decoder must store sufficient information to make it possible to restore exactly the original decoded and unfiltered pixel samples (but not necessarily the original compressed datastream or line-by-line filter selections), and data from any additional recognized PNG chunks such as PLTE, gAMA, and tRNS.
- An *abstract* image can be stored by the decoder in any form that is convenient, such as an X Windowing System "pixmap", even though that form might not have sufficent resolution for exact, lossless conversion. In the case of a PNG image, the pixels could be stored after the gamma and chromaticity corrections have been made, and the sample depth could be the same as the display hardware, even though it is smaller than the original sample depth.

MNG is pronounced "Ming."

When a MNG datastream is stored in a file, it is recommended that ".mng" be used as the file suffix. In network applications, the Media Type "image/x-mng" can be used. Registration of the media type "image/mng" might be pursued at some future date.

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.

MNG does not yet accommodate sound or complex sequencing information, but 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 a lossy image format such as the proposed PNP (Portable Network Photo) format might be added. PNP is under discussion by pnp-list@dworkin.wustl.edu.

Chunk structure (length, name, data, 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.

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 (in this document a "sprite" is an object that has location and visibility information. We do not imply that the object is automatically redrawn whenever it changes; explicit directives to redraw a sprite are required).

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.

Because the images making up a MNG are in PNG format, MNG shares the good features of PNG:

- It is unencumbered by patents.
- It is streamable.
- It has excellent, lossless compression.
- It stores up to four channels (red, green, blue, alpha), with up to 16 bits per channel.
- It provides transparency and an alpha channel.
- It provides platform-independent rendition of colors by inclusion of gamma and chromaticity information.
- It provides early detection of common file transmission errors and robust detection of file corruption.
- Single-image GIF files can be losslessly converted to PNG.

In addition,

- It provides animation with variable interframe delays.
- It allows composition of frames containing multiple images.
- It facilitates the use of images as "sprites" that can be reused in subsequent frames.
- It capitalizes on frame-to-frame similarities to reduce the amount of data that must be included in the datastream.
- It provides "restart" points at which an animation can be resumed in case of data loss or corruption.
- A "frame priority" chunk allows authors to indicate which frame should be displayed by single-image viewers, and a subset of the frames that should be displayed by slow viewers.
- An "eXPI" chunk allows authors to mark one or more images for export outside the scope of MNG, where they can be used for icons or similar purposes.
- Multiple-image GIF files can be losslessly converted to MNG.

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 12 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: ticks_per_second  (unsigned integer).
         This is the unit used by the FRAM chunk to
         specify frame_duration and sync_timeout.
```

The "max_frame_width" and "max_frame_height" fields give the intended display size (measured in pixels) and provide default clipping boundaries (see Paragraph 2.3.2, below). These can be set to zero if the MNG datastream contains no visible images.

The "ticks_per_second" field must be nonzero if the datastream contains more than one frame. It should be set to zero if the datastream contains exactly one frame, so that decoders and other applications like the UNIX "file" command can use this field to identify single-frame MNGs.

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 FREZ Freeze information

The "FREZ" chunk marks a point in the datastream at which images and other chunk information are "frozen"; decoders encountering a "SEEK" can rely upon this information (and only this information) being available.

The "FREZ" chunk is empty.

It appears after the set of chunks that define information that must be retained for the remainder of the datastream. The "FREZ" chunk must be present when the "SEEK" chunk is present. These can be chunks that define images, or they can be chunks that define other information such as "gAMA", "cHRM", and "sPLT".

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

It is not permitted, at any point beyond the "FREZ" chunk, to modify any image or other chunk data that appeared ahead of the "FREZ" chunk. An image appearing ahead of the "FREZ" chunk can later be discarded by naming its "image_id" in a "DISC" chunk; if so, its "image_id" must not be reused.

An image appearing ahead of the "FREZ" can be the subject of a "CLON" chunk. If the clone is a partial clone, modifying it is not permitted, because this would also modify the original image.

In the case of chunks like "gAMA", which overwrite a current value, the chunk is not permitted after the "FREZ" chunk if it has appeared ahead of it. In the case of chunks like "sPLT", which can occur multiple times, with a different "purpose" field, additional instances of the chunk are permitted after the "FREZ" chunk, but not with the same keyword as any instances that occurred ahead of the "FREZ" chunk.

Only one instance of the "FREZ" chunk is permitted in a MNG datastream, and, if the "SEEK" chunk is present, the "FREZ" chunk must be present, prior to the first "SEEK" chunk.

2.1.4 SEEK Seek point

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

The "SEEK" can be empty, or it can contain a keyword.

```
n bytes:  keyword (Latin-1 string).
```

The keyword is optional. It 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 keyword. Keywords are case-sensitive. Use caution when printing or displaying keywords (Refer to Security considerations, Chapter 6, below). No specific use for the keyword is specified in this document, but applications can use the keyword for such purposes as constructing a menu of SEEK points for a slide-show viewer.

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

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

When the "SEEK" chunk is encountered, the decoder can discard any image data appearing after the "FREZ" chunk, as though an empty "DISC" chunk were present.

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.

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 "FREZ" chunk. The "FREZ" chunk must also be present if the "SEEK" chunk is present.

2.1.5 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: must_run_loop (unsigned integer).
      0: The decoder must run the loop exactly
         repeat_count times, because execution of
         the loop modifies or relocates basis
         images that might be needed later.
      1: It is safe for a decoder to run the loop
         fewer than repeat_count times, provided
         that it runs the loop at least once. Every
         iteration of the loop generates an
         identical sequence of frames.
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 "must_run_loop" 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 "must_run_loop" field are true. When "must_run_loop=1", all iterations of the loop are identical, and a viewer could choose to actually execute the loop just once, while storing copies of the composited frames for redisplay, and then replaying the stored images for the remaining "repeat_count" iterations. Also, when "must_run_loop=1", it is safe to escape from the loop upon reaching the "ENDL" chunk even though the loop has not yet been executed for "repeat_count" iterations. It is unsafe, however, to escape from the interior of a loop while it is being executed; in this event, the decoder must proceed to the next "SEEK" chunk before resuming.

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 Define an image

The "DEFI" chunk sets the default image_id, visibility, abstract_flag, location, and clipping boundaries for any subsequent images that are defined with "IHDR-IEND" or "BASI-IEND" datastreams. The "DEFI" chunk contains 2, 3, 4, 12, or 28 bytes:

```
2 bytes: image_id (unsigned integer) image identifier
         to be given to the images that follow the
         DEFI chunk.

1 byte:  visibility (unsigned integer)
     0:  Make the images visible.
     1:  Make the images invisible.
     This field can be omitted if the abstract_flag,
     location, and clipping boundary fields
     are also omitted.  When it is omitted, the image
     is made visible (visibility=0).
```

```
1 byte:  abstract_flag (unsigned integer)
     0:  Make the images "abstract" (image can not
         be the source for a PND)
     1:  Make the images "concrete" (image can be
         modified by a PND).
     This field can be omitted if location, and
     clipping boundary fields are also omitted.
     When it is omitted, the image is made "abstract"
     (abstract_flag=0).

4 bytes: x_location (signed integer).
     The x_location and y_location fields can be
     omitted if the clipping boundaries are also
     omitted.  If so, decoders must assume default
     values {x_location,y_location}={0,0}.

4 bytes: y_location (signed integer).

4 bytes: left_cb (signed integer). Left clipping
     boundary.  The left_cb, right_cb, top_cb, and
     bottom_cb fields can be omitted as a group.
     If so, decoders must assume default values
     {0, max_frame_width, 0, max_frame_height}.

4 bytes: right_cb (signed integer).

4 bytes: top_cb (signed integer).

4 bytes: bottom_cb (signed integer).
```

If the "image_id" for an image is nonzero, subsequent "DHDR", "SHOW", "CLON", "LOCA", "CLIP", and "DISC" chunks can use this number to identify it. This also is used as the level, or position in the stacking order, of the image in a composite frame. The image having the smallest value of "image_id" is on the bottom (it can be obscured by any other "image_id"), while the image having the largest value is the top image (it cannot be obscured by any other "image_id").

When the "image_id" for an image is zero, it can be discarded immediately after it has been processed, and it can be treated as an "abstract" image, regardless of the contents of the "abstract_flag" field.

Negative values are permitted for the X and Y image location and clipping boundaries. The positive directions are downward and rightward from the upper left corner of the display.

The "DEFI" chunk values remain in effect until another "DEFI" chunk or a "SEEK" chunk appears. If no "DEFI" chunk is in effect, the default values for the parameters are

```
      image_id := 0
   visibility := 0
abstract_flag := 0
   x_location := 0
   y_location := 0
      left_cb := 0
     right_cb := max_frame_width
       top_cb := 0
    bottom_cb := max_frame_height
```

If "image_id" is an identifier that already exists when an "DEFI", "IHDR" or "BASI" chunk is encountered, the basis image previously associated with the identifier is discarded (or disassociated with it, if the identifier was a partial clone of another image), along with any "LOCA", "CLIP", "INHR", or "SBYK" data associated with it.

If no "DEFI" chunk is in effect (either because there is none in the datastream, or because a "SEEK" chunk has caused it to be discarded), the decoder must use the following default values:

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.

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

If "visibility=0" for the image when the "IHDR" chunk is encountered, a viewer can choose to display the image while it is being decoded, perhaps taking advantage of the PNG interlacing method.

If "image_id=0", there is no need to store the pixel data after displaying it.

If "abstract_flag=1" and "image_id != 0", the decoder must store the original pixel data losslessly, because it is possible that a subsequent PND stream might want to modify it. If "abstract_flag=0", the decoder can store the pixel data in any form that it chooses.

2.2.3 BASI, PNG chunks, IEND

The first 13 bytes of the "BASI" chunk are identical to those of the "IHDR" chunk, and an optional 8 additional bytes provide 16-bit {red, green, blue, alpha} values, that are used to fill the entire generated image when the "IDAT" chunk is not present.

```
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).
2 bytes: red sample (unsigned integer).
2 bytes: green sample (unsigned integer)..
2 bytes: blue sample (unsigned integer).
2 bytes: alpha sample (unsigned integer).
```

The alpha sample may be omitted. If so, and the "color_type" is one that requires alpha, the alpha value corresponding to an opaque pixel will be used. If the color samples are omitted, zeroes will be used. The decoder is responsible for converting the color and alpha samples to the appropriate format and sample depth for the specified "color_type". When "color_type=3", the decoder must generate a palette of length 2^bit_depth, whose first entry contains the given {red, green, blue} triple, and whose remaining entries are filled with zeroes.

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, or it can be used to introduce a simple blank or colored rectangle into which other images will be pasted by means of the "PAST" chunk.

A "BASI" chunk appearing in a MNG datastream must be preceded by a "DEFI" chunk that gives a nonzero "image_id", the location for the basis image, and "visibility=1". The "abstract_flag" can be either 0 (abstract) or 1 (concrete), depending on whether the basis image is intended for subsequent use by a PND datastream or not.

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, and the resulting image is displayed if its "visibility=0". The image width and height must not exceed "max_image_width" and "max_image_height" from the "MHDR" chunk. The image must have "abstract_flag=1"

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 4, 5, 6, 7, or 16 bytes.

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

2 bytes: clone_id (nonzero 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 and CLIP data and image
         data.

      1: partial clone; only the LOCA and CLIP data is
         copied and a link is made to the image data.

      2: renumber image (this is equivalent to
         "CLON old_id new_id 1 DISC old_id").  This
         can be useful for rearranging the stacking
         order of images in a composite frame.

      This field can be omitted if the "visibility"
      field is also omitted.  If it is omitted, the
      clone_type defaults to zero (full clone).
```

```
1 byte:  visibility (unsigned integer)
     0:  Make the clone visible.
     1:  Make the clone invisible.
     This field can be omitted if the abstract_flag and
     location fields are also omitted.  When it is
     omitted, the image is made visible (visibility=0).

1 byte:  abstract_flag (unsigned integer)
     0:  abstract_flag is the same as that of the basis
         image.
     1:  Make the clone "abstract" (abstract_flag=0).

     This field can be omitted if the location fields
     are also omitted.  When it is omitted, the image
     retains the abstract_flag of the basis image.

1  byte: loca_delta_type (unsigned integer)

       0: LOCA data gives x_location and
          y_location directly.

       1: New positions are determined by adding
          the LOCA data to the position of the
          basis image.

       This field, together with the x_location and
       y_location fields, can be omitted.  When they
       are omitted, decoders must assume default
       values {x_location,y_location}={0,0}.

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 from the upper left corner of the display.

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

Subsequent chunks can modify, show, or discard a full clone or modify its associated "LOCA", "CLIP", "INHR", or "SBYK" data without affecting the basis image, or they can modify, show, or discard the basis image or modify 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" or "CLIP" data can be changed independently for each partial clone.

2.2.6 PAST Paste an image into an another

Paste an image or images identified by "source_id", or part of it, into an existing abstract image identified by "destination_id".

The "PAST" chunk contains a 2-byte "destination_id" and 9 bytes giving a "target location", plus one or more 30-byte sequences.

```
2 bytes: destination_id (unsigned integer)

1 byte:  target_delta_type (unsigned integer)
     0:  target_x and target_y are given directly
     1:  target_x and target_y are deltas from
         their previous values

4 bytes: target_x (signed integer)

4 bytes: target_y (signed integer)

2 bytes: source_id (unsigned nonzero integer)
         an image to be pasted in.

1 byte: composition_mode (unsigned integer)
     0: replace
     1: composite_over
     2: composite_under

1 byte: orientation (unsigned integer)
     0: same as source image
     2: flipped left-right, then up-down
     4: flipped left-right
     6: flipped up-down

1 byte:  offset_delta_type (unsigned integer)
     0:  offsets are measured from {0,0}
     1:  offsets are measured from
         {target_x, target_y}
4 bytes: xoffset or delta xoffset (signed integer)
4 bytes: yoffset or delta yoffset (signed integer)
```

```
1 byte:  boundary_delta_type (unsigned integer)
     0:  boundaries are measured from {0,0}
     1:  boundaries are measured from
         {target_x, target_y}
4 bytes: left_pb or delta left_pb (signed integer)
4 bytes: right_pb or delta right_pb (signed integer)
4 bytes: top_pb or delta top_pb (signed integer)
4 bytes: bottom_pb or delta bottom_pb (signed integer)
etc.
```

The destination image must have the "abstract" property `(abstract_flag=0)`. The source images can be "abstract" or "concrete" and have any "`color_type`" and "`sample_depth`". The number of source images is `((chunk_length-11)/30)`.

The "`xoffset`" and "`yoffset`" distances and the clipping boundaries are measured, in pixels, positive rightward and downward from either the {`0,0`} position or the {`target_x, target_y`} position in the destination image. They do not necessarily have to fall within the destination image. Only those pixels of the source image that fall within the destination image and also within the specified clipping boundaries will be copied into the destination image. If the source image has been flipped or rotated, "`xoffset`" and "`yoffset`" give the location of its new upper lefthand corner.

When "`composition_mode=1`" any non-opaque pixels in the source image are combined with those of the destination image (if the destination pixel is also non-opaque, the resulting pixel will be non-opaque)

When "`composition_mode=2`" any non-opaque pixels in the destination image are combined with those of the source image (if the source pixel is also non-opaque, the resulting pixel will be non-opaque)

The order of composition is the same as the order that the "`source_id's`" appear in the list (but a decoder can do the composition in any order it pleases, provided that the resulting destination image is the same as if it had actually performed each composition in the specified order.

The "`LOCA`" or "`CLIP`" information associated with the "`destination_id`" and the "`source_id's`" is not used in the "`PAST`" operation (but if a decoder is simultaneously updating and displaying the "`destination_id`", the "`LOCA`" and "`CLIP`" for the "`destination_id`" is used in the display operation).

2.2.7 `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 (nonzero 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 "FREZ" 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", "CLIP" "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 "FREZ" 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" or "CLIP" data can be discarded. The image data must be retained until the last remaining partial clone is discarded.

2.3 Critical MNG image displaying chunks

2.3.1 LOCA Image location

Location of an existing 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 (nonzero unsigned integer).

    1  byte: loca_delta_type (unsigned integer).
          0: LOCA data gives x_location and
             y_location directly.
          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 2.3.2, below.

It is an error for the "LOCA" chunk to appear with an "image_id" that has not previously been defined.

2.3.2 CLIP Image clipping boundaries

This chunk gives the boundaries to which an existing image must be clipped for display. It contains the following 19 bytes:

```
2 bytes: image_id (nonzero unsigned integer).

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
         for this image_id.

4 bytes: left_cb or delta_left_cb (signed integer).

4 bytes: right_cb or delta_right_cb (signed integer).

4 bytes: top_cb or delta_top_cb (signed integer).

4 bytes: bottom_cb or delta_bottom_cb (signed
         integer).
```

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 all of the following are true:

```
0       <= x < max_frame_width   (from the MHDR chunk)
0       <= y < max_frame_height
left_fb <= x < right_fb          (from the FRAM chunk)
top_fb  <= y < bottom_fb
left_cb <= x < right_cb          (from the CLIP chunk)
top_cb  <= y < bottom_cb
```

It is an error for the "CLIP" chunk to appear with an "image_id" that has not previously been defined. When an "image_id" is discarded, its "CLIP" data is also discarded.

2.3.3 FRAM Delimit a frame

The "FRAM" chunk marks the beginning of a group of chunks that describe a frame. 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 delimiter. A nonempty one is a frame delimiter, and it also changes "FRAM" parameters, either for the upcoming frame or until reset. When the "FRAM" chunk is not empty, it contains five 1-byte fields plus a variable number of optional 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.  The level, or stacking
         order, of each image is given by its
         image_id.

     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.  The level, or stacking order,
         of each image is given by its image_id.
```

```
          4:  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).

          5:  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_frame_boundaries.
          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.

     1  byte: fb_delta_type (unsigned integer).
           (omit this and the following four fields
           if change_frame_boundaries=0).
           0: frame_boundary values are given directly.
           1: frame_boundaries are determined by adding
              the FRAM data to their previous values.

     4 bytes: left_fb or delta_left_fb (signed integer).

     4 bytes: right_fb or delta_right_fb (signed integer).

     4 bytes: top_fb or delta_top_fb (signed integer).

     4 bytes: bottom_fb or delta_bottom_fb (signed
              integer).

     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 being 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.

The initial values of the "FRAM" parameters are

```
framing_mode := 1
frame_duration := 0
left_fb   := 0
right_fb  := max_frame_width
top_fb    := 0
bottom_fb := max_frame_height
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. The "CLIP" chunk can be used to specify clipping boundaries for each image. 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 or is not recognized by the decoder.

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 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. The "frame_duration" value is the desired minimum time to elapse from the beginning of displaying one image until the beginning of displaying the next image. The following events trigger the display of a frame:

- Decoding a "IHDR-IEND" sequence at the MNG level, when it defines a visible image.

- Decoding a "DHDR-DEND" sequence, when it defines a visible image.
- Decoding a "BASI-DEND" sequence, when it defines a visible image.
- Decoding a "PAST" chunk, when it defines a visible image.
- Decoding a "SHOW" chunk, when it makes an image visible. When the "SHOW" chunk makes multiple images visible, each one in turn is a separate frame.

For example, the sequence

```
FRAM 1
SHOW 1 5
FRAM
```

will result in the display of five frames, each containing one of the images against the background according to its "LOCA" and "CLIP" information.

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. For example, the sequence

```
FRAM 2
SHOW 1 5
FRAM
```

will result in a single frame containing five images, each displayed according to its "LOCA" and "CLIP" data.

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 must be given a unique "image_id", which defines its position in the stacking order. The image with the smallest value of "image_id" is the background image (which can be obscured by any other image) and the image with the largest values of "image_id" is 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. They must be made visible in ascending value of "image_id". If "LOCA" or "CLIP" 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", 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.

When multiple images are made visible with a single "SHOW" chunk, a frame boundary occurs after each one. For example, the sequence

```
FRAM 3
SHOW 1 5
FRAM
```

would result in five frames being displayed, the first with only image 1, the second with images 1 and 2, etc., and the fifth frame with all five images visible. The resulting output is exactly equivalent to what would have been accomplished (but perhaps less efficiently) with

```
FRAM 2
SHOW 1
FRAM
SHOW 1 2
FRAM
SHOW 1 2 3
FRAM
SHOW 1 2 3 4
FRAM
SHOW 1 2 3 4 5
FRAM
```

If "LOCA" or "CLIP" 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", "draw mode" decoders can treat the frame exactly like one with "framing_mode=2".

But, because different images having the same "image_id" might be present in the frame, "sprite mode" decoders will have to revert to "draw mode" to display this frame type, or or else they will have to make their 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=4" for storage of things like tomographic slices.

If "LOCA" or "CLIP" chunks appear, they must appear before the images to which they pertain are made visible.

When "framing_mode=5", "draw mode" decoders can treat the frame in the same way as when "framing_mode=3". If "LOCA" or "CLIP" chunks appear, they must appear before the images to which they pertain are made visible. "Sprite mode" decoders will have to revert to "draw mode" to display this frame type, or or else they will have to make their 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 (or between images, when "framing_mode=1"). It is measured in "ticks" using the tick length determined from "ticks_per_second" defined in the "MHDR" chunk.

The value of "frame_duration" will remain in effect until another "FRAM" chunk redefines it.

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 0 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.

2.3.4 SHOW Show images

The "SHOW" chunk can be used to show or hide one or more previously-defined images. It contains 2, 4, or 5 bytes:

```
2 bytes: first_id (nonzero unsigned integer).

2 bytes: last_id  (nonzero unsigned integer).
     This field can be omitted if the visibility
     byte is also omitted.  Decoders must assume
     the default value, "last_id=first_id."

1 byte:  visibility (unsigned integer)
     0:  Make the images visible.
     1:  Make the images invisible.
     This field can be omitted.  If so, decoders
     must assume the default, "visibility=0".
```

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, because "sprite mode" viewers would show them in normal order anyway). 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 and the images have already been made visibile in the frame, because "draw mode" viewers will have already drawn them and have no way to make them invisible again without redrawing the entire frame).

An instance of each image will be displayed at the location specified by the "DEFI", "CLON", or "LOCA" chunk and clipped according to the boundaries specified by the "CLIP" chunk. When the "LOCA" or "CLON" 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" or "CLON" chunk.

It is not necessary to follow an "IHDR–IEND" or "DHDR–DEND" sequence with a "SHOW" chunk, if its visibility is the same as that given in the "DEFI" chunk that introduced the "image_id".

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 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, clipped to the frame boundaries given in the FRAM chunk (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 (but if one appears before the "FREZ" chunk, then none is permitted after it).

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.

The "bACk" components are always interpreted in linear color space (gamma=1), regardless of the presence of any "gAMA" chunk in the datastream.

2.4.2 eXPI Export image

The "eXPI" chunk takes a snapshot of an image (either concrete or abstract), associates the name with that snapshot, and makes the name available to the "outside world" (like a scripting language).

The chunk contains an image identifier and a name:

```
2 bytes: snapshot_id (unsigned nonzero integer)
n bytes: snapshot_name (Latin-1 text)
```

Note that the "snapshot_name" is associated with the snapshot, not with the "snapshot_id" nor its future contents; discarding the image identified by "snapshot_id" will not affect the snapshot. The "snapshot_name" means nothing inside the scope of the MNG specification. If two "eXPI" chunks use the same name, it's the outside world's problem (and the outside world's prerogative to regard it as an error). A decoder that knows of no "outside world" can simply ignore the "eXPI" chunk. This chunk could be used in MNG datastreams that define libraries of related images, rather than animations.

Names beginning with the word "thumbnail" are reserved for snapshot images that are intended to make good icons for the MNG. Thumbnail images are regular PNG or PND images, but they would normally have smaller dimensions and fewer colors than the MNG frames. They can be defined with the visibility field set to "invisible" if they are not intended to be shown as a part of the regular display.

Multiple instances of the "eXPI" chunk are permitted in a MNG datastream.

2.4.3 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: Priority is given directly.
     1: Priority is determined by adding
        the fPRi data to the previous
        values, modulo 256.
```

```
1 byte: priority or delta_priority (signed
        integer). Value to be assigned to
        subsequent frames until another fPRi
        chunk is reached.

1 byte: max_priority (unsigned integer).  Maximum
        priority that will appear in any subsequent
        fPRI chunk (max_priority is always given
        directly without regard to fPRi_delta_type).
```

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 5 and 8.

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.

Multiple instances of the "fPRi" chunk are permitted. If the "FREZ" chunk is present, the "fPRI" should not precede it, because it would then become impossible to change the priority values later in the datastream.

2.4.4 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 private critical 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 private critical chunk names, but they could be any string that a decoder is required to recognize. No critical chunks defined in this specification or in the PNG specification should be named in a "nEED" chunk, because MNG-compliant decoders are required to recognize all of

them, whether they appear in a "nEED" chunk or not. The purpose of the "nEED" chunk is only to identify requirements that are above and beyond the requirements of this document and of the PNG specification.

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. A null separator byte must appear after each keyword in the "nEED" chunk except for the last one.

Decoders that do not recognize 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.

[During the draft phase of the development of this specification, you can specify that the datastream is written in accordance with a certain draft version of MNG, you can use "nEEDdraft nn"].

2.4.5 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"

A MNG editor that writes PNG datastreams should not include the top-level "tEXt", "tIME", and "zTXt" chunks in the generated PNG datastreams.

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 subsequent PNG datastreams:

- "cHRM"
- "gAMA"
- "sRGB"

A MNG editor that writes PNG datastreams is expected to include the top-level "cHRM", "gAMA", and "sRGB" chunks in the generated PNG datastreams.

The following PNG chunks are also defined at the MNG top level. They provides values that take precedence over those that might be provided in subsequent PNG datastreams and provide values to be used when they are not provided in subsequent PNG datastreams:

- "pHYs" If the "pHYs" chunk is present in any subimage of a composite frame, a "pHYs" chunk must also be present at the MNG top level. This ensures that viewers will only have one set of "pHYs" values to deal with while displaying a composite frame, and any scaling that is necessary can be applied to the entire composite frame and not to its separate images. When a frame is comprised only of a single

subimage that fills the entire frame, or when all subimages of the frame fill the entire frame and have the same "pHYs" values, the top-level "pHYs" chunk can be omitted.

- "sPLT" (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"). This chunk can appear for any color type. There can be multiple "sPLT" 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 "FREZ" chunk with the same "palette_name" as one that appears ahead of the "FREZ" chunk. It is permitted, however, to define and redefine other palettes with other "palette_name" fields.

  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 "sPLT" 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 datastreams should ignore the "sPLT" and "pHYs" data from the MNG level and simply copy any "sPLT" and "pHYs" chunks appearing within the PNG datastreams.

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 "BASI" chunk with the appropriate dimensions and an "IEND" chunk, 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 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, and displays it if its "visibility=0").

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 (nonzero 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_id" must identify an existing image, and the image must be a "concrete" image, i.e. it must have the property "abstract_flag=1".

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.

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.
1 byte: fill_method
     0: left-bit-replication
     1: zero fill
```

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.

When the new bit_depth exceeds the original bit_depth, the additional bits of each sample must be filled according to the specified "fill_method". See the PNG specification [LINK] for further information on these filling methods.

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.

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.

If the "PLTE" chunk is present, it need not have the same length as that inherited from the basis image, but it must contain the complete palette needed in the generated image. If it is shorter than the palette of the basis image, decoders can discard the remaining entries. They can also truncate any "tRNS" data inherited from an indexed-color basis image. If it is longer, and a new "tRNS" chunk is not present in an indexed-color image, the "tRNS" data should be extended with opaque entries.

[What about defining an IPLT chunk that can give an incomplete PLTE, that would only overwrite palette entries without truncating the resulting PLTE? This might be useful for palette-animation applications.]

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 "oFFs" chunk. MNG editors are expected 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.
```

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.

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" and "IDAT" (both with "rule=0") 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, 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:  chunkname (four ASCII bytes).
n bytes:  keyword (Latin-1 string).
```

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 keyword. Keywords are case-sensitive. Use caution when printing or displaying keywords (Refer to Security considerations, Chapter 6, 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 only be used to select PNG chunks whose keyword, "purpose," or other identifying string appears *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.

When the "INHR" chunk is also present, and the decoder is not ignoring it, the "INHR" chunk data must be processed first. Note that if a chunk name appears in the "SBYK" chunk and also appears in the "INHR" chunk with "rule=2", there will be nothing left for "SBYK" to select from.

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

4 Recommendations for encoders

Say something useful here.

5 Recommendations for decoders

Say something useful here.

5.1 Retaining image data

The decoder must retain information about each image (except for images with "image_id=0") 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 nonzero "image_id" that is defined and not subsequently discarded:

- The data from the "LOCA" and "CLIP" chunks.
- 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 "image_id=0" or by using the "DISC" or the "SEEK" chunk.

5.2 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 "FREZ" 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 datastream with errors unless the user intervenes.

5.3 Decoder handling of interlaced images

Decoders are required to be able to interpret datastreams that contain interlaced PNG images, 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.

5.4 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.

6 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 "SEEK" chunk makes it safe for a viewer to resume processing after it encounters a corrupted portion of a MNG datastream).

No known additional security concerns are raised by this format.

7 Appendix: Examples

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

7.1 Example 1: A single image

The simplest MNG datastream is a single-image PNG datastream. The simplest way to create a MNG from a PNG is:

```
copy file.png file.mng
```

The resulting MNG file looks like:

```
\211 P N G \r \n ^z \n  # PNG signature
IHDR 720 468 8 0 0 0 0  # Width and height, etc.
IDAT ...
IEND
```

7.2 Example 2: Very simple movie

This example demonstrates a very simple movie, such as might result from directly converting an animated GIF that contains a simple series of full-frame images:

```
MHDR 256 300   # Width and height
     30        # 30 ticks per second

FRAM 1 2 0 0 0 30 # set frame_duration to 1 sec

DEFI 1 IHDR ...  IDAT ...  IEND
DEFI 2 IHDR ...  IDAT ...  IEND
DEFI 3 IHDR ...  IDAT ...  IEND
DEFI 4 IHDR ...  IDAT ...  IEND
DEFI 5 IHDR ...  IDAT ...  IEND
DEFI 6 IHDR ...  IDAT ...  IEND
DEFI 7 IHDR ...  IDAT ...  IEND
DEFI 8 IHDR ...  IDAT ...  IEND

LOOP 0 0 0
SHOW 1 8
ENDL 0
MEND
```

7.3 Example 3: Simple slideshow

A simple slideshow:

```
\212 M N G \r \n ^z \n  # MNG signature
MHDR 720 468   # Width and height
     1         # 1 tick per second
FRAM 1 2 2 0 2 0 600 0 # set frame_duration to 0 sec
  # sync_timeout to 600 sec and sync_id list to {0}
FREZ
SEEK "Briefing to the Workforce"
IHDR ...  IDAT ...  IEND  # DEFI 0 is implied
SEEK "Outline"
IHDR ...  IDAT ...  IEND
SEEK "Our Vision"
IHDR ...  IDAT ...  IEND
SEEK "Our Mission"
IHDR ...  IDAT ...  IEND
SEEK "Downsizing Plans"
IHDR ...  IDAT ...  IEND
MEND
```

7.4 Example 4: A more efficient slideshow

This slideshow gives exactly the same output as Example 3, but the storage in the datastream is more efficient (the IDAT chunks will be smaller) while the memory requirements in the decoder are larger. Image ID 1 is used to store the ornate logos and frame design that is found on every slide. The individual frames only contain the text and other information that is unique to each slide.

```
\212 M N G \r \n ^z \n  # MNG signature
MHDR 720 468   # Width and height
     1         # 1 tick per second
DEFI 1 1       # define image 1, invisible, abstract
IHDR ...  IDAT ...  IEND
FRAM 2 2 2 0 2 0 600 0 # set frame_duration to 0 sec
  # sync_timeout to 600 sec and sync_id list to {0}
FREZ
SEEK "Briefing to the Workforce"
SHOW 2
SEEK "Outline"
CLON 1 2 DHDR 2 ...  IDAT ...  IEND SHOW 2 FRAM
SEEK "Our Vision"
CLON 1 2 DHDR 2 ...  IDAT ...  IEND SHOW 2 FRAM
SEEK "Our Mission"
CLON 1 2 DHDR 2 ...  IDAT ...  IEND SHOW 2 FRAM
SEEK "Downsizing Plans"
CLON 1 2 DHDR 2 ...  IDAT ...  IEND SHOW 2 FRAM
MEND
```

7.5 Example 5: Simple movie

This movie is still fairly simple, but it capitalizes on frame-to-frame similarities by use of PND datastreams, and also demonstrates the use of the "fPRi" chunk.

```
\212 M N G \r \n ^z \n  # MNG signature
MHDR 720 468   # Width and height
     30        # 30 ticks per second
tEXtTitle\0Sample Movie
fPRI 128 255   # default frame priority is "medium"
FRAM 1 2 0 0 0 3 # set frame_duration to 1/10 sec
DEFI 1 0 1       # set default image_id to 1 (concrete)
FREZ
SEEK "start"

IHDR 720 468 8 2 0 0 0   # DEFI 1 is implied
IDAT ...
IEND

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

DHDR 1 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 "frame 3"  # 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 1 1 1 720 468 0 0     # Another PNG-delta frame
IDAT ...   # The entire 720x468 rectangle changes
DEND       # this time.

SEEK "end"
MEND            # End of MNG datastream
```

7.6 Example 6: Single composite frame

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

```
\212 M N G \r \n ^z \n # MNG signature
MHDR 1024 512 # Width, height
     1        # ticklength
bACk 1 16448 16448 52800     # Must use sky blue background

DEFI 1 1     # Define invisible abstract thumbnail image.
IHDR 64 64 4 3 0 0 0
IDAT
IEND
eXPI 1 "thumbnail 1"
DEFI 1 1       # Also define a larger thumbnail.
IHDR 96 96 4 3 0 0 0
IDAT
IEND
eXPI 1 "thumbnail 2"
DISC           # Discard the thumbnail image.

FRAM 2 0 0 0 0       # Composite frame, 1024 x 512
DEFI 1 0 0 1 6 6     # Define first (bottom) image
IHDR 500 500 16 0 .. # A 16-bit graylevel image
gAMA 50000
IDAT ...
IEND                 # End of image

CLON 1 2 0 0 1 0 518 6 # Make a full "concrete" clone
DHDR 2 1 5           #  Modify it (no change to pixels).
INHR gAMA 0 tEXt 0 faLT 0 IDAT 0 # 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
DEFI 3 0 0 1 900 400 # Overlay near lower right-hand corner
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
```

7.7 Example 7: Movie with sprites

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

```
\212 M N G \r \n ^z \n  # MNG signature
MHDR 512 512     # Start of MNG datastream
     30          # ticklength
FRAM 2 2 0 0 0 3 # First frame; set frame_duration=3 ticks
DEFI 1           # Define image 1 (abstract, LOCA 0 0)
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 0 1       # Define image 2, concrete
LOCA 2 0 300 200   # Location for image 2
IHDR 32 32 ...   # It's a small PNG
gAMA 50000
IDAT ...
IEND
FRAM             # Start new frame
                 # New location for image 1 is still 0,0
SHOW 1           # Display image 1 from previous frame
LOCA 2 1 10 5    # New (delta) location for image 2
SHOW 2             # Retrieve image 2 from previous frame,
CLON 2 3 0 1     # make a full clone of it as image 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 3 is implied by CLON visibility
FRAM             # Next frame (repeat this FRAM-SHOW 1 3
                 #   sequence with different locations to
                 #   move the images around)
                 # New location for image 1 is still 0,0
LOCA 2 1 10 5    # New (delta) location for image 2
LOCA 3 1 5 -2    # New location for image 3
SHOW 1 3         # Show images 1 through 3
FRAM             #  Another frame
etc.
FRAM
etc.             # More frames
MEND             # End of MNG datastream
```

7.8 Example 8: "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.

```
\212 M N G \r \n ^z \n # MNG signature
MHDR 64 64        # Width, height
     30           # Tick length
bACk 0 52800 52800 52800 # "Browser gray" default background

FRAM 1 2 0 0 0 3 # First frame; set frame_duration=3 ticks
DEFI 1 0 1  # visible and "concrete"
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    # 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 0 60 # Hold the last frame for at least 60
      # ticks (2 sec).  Applications might show it longer.
SHOW 1      # This copy still has alpha=255 for the
            # opaque pixels and alpha=0 for the others.
MEND        # End of MNG
```

7.9 Example 9: 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.

```
\212 M N G \r \n ^z \n # MNG signature
MHDR 150 150     # Width, height
     1           # Tick length (unused in this example)

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 and pCAL
 specification version 19961230 available at
 ftp://swrinde.nde.swri.edu/pub/png-group/documents/
xsCL x\0 [sig] kilometers\0 0\0 150
ysCL x\0 [sig] kilometers\0 0\0 150
zsCA x\0 [sig] Height (kilometers)\0 0\0 15
tsCL x\0 [sig] Time (hours)\0 0\0 24
pcAL x\0 [sig] 0 255 0 2 Degrees Celsius\0 0\0 45
DEFI 1 0 1       # all images will have image_id = 1
FREZ             # and be visible and "concrete"
SEEK
FRAM 4 0 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 1 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
FRAM         # End of first block
etc.         # Repeat FRAM through SEEK 19 more times
SEEK
MEND         # End of MNG
```

7.10 Example 10: 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:

```
\212 M N G \r \n ^z \n  # MNG signature
MHDR 1024 768 1 # Start of MNG datastream
FRAM 4 0 0 0 0
DEFI 1 1 0 0 -64 # Set up an offscreen "abstract" copy
IHDR 128 64 ...  # of the tile
PLTE ...
IDAT ...      # Nothing will be displayed because it's
IEND ...      # outside the 1024 by 768 composite frame
LOOP 0 0 12   # Y loop -- make 12 rows of tiles
LOCA 1 1 0 64 # Move the first copy down 64 rows
SHOW 1        # Display it
CLON 1 2      # 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 2        # Use the second copy
ENDL 1
ENDL 0
MEND
```

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

```
\212 M N G \r \n ^z \n  # MNG signature
MHDR 1024 768 1 # Start of MNG datastream
FRAM 3 0 0 0 0
DEFI 1 1       # Corner tile, image_id=1, invisible,
IHDR 128 64 ... # "abstract" (clones will be abstract)
PLTE ...
IDAT ...
IEND ...
CLON 1 2 1 0 1 0 128 # Make partial clones, delta located
CLON 2 3 1 0 1 0 128 # Locate each one 128 columns to the
CLON 3 4 1 0 1 0 128 # right of its basis image
etc.
CLON 1 13 1 0 1 0 64  # Make a partial clone below image 1
CLON 13 14 1 0 1 0 128 # More clones to the right
etc.
SHOW 1 96      # Show the image and its partial clones
MEND
```

7.11 Example 11: 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.

```
\212 M N G \r \n ^z \n  # MNG signature
MHDR 512 256      # Width and height on screen
     30          # Tick length
FRAM 2 2 0 0 0 1 # frame_duration = 1 tick
DEFI 1 1 0 0 256  # Define image 1 but don't display now
                  # 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
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
             # It ends up just above the 512 by 256 window
             # The border does not move
SHOW 1 2     # Show the two images
ENDL 0

MEND
```

7.12 Example 12: Converting a GIF animation to MNG

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

```
begin
    write "MHDR" chunk
    write "BACK" chunk
    saved_images := 0
    frame_duration := 0
    first_frame := TRUE
    if(loops>1) "write LOOP 0 1 loops"
    for subimage in gif89a file do
       if(frame_duration != gif_duration) then
         frame_duration := gif_duration
         write "FRAM 2 2 2 0 2 0 frame_duration 0" chunk
         first_frame := FALSE
       else if(first_frame == TRUE)then
         write "FRAM 2 0 0 0 0" chunk
         first_frame := FALSE
       else
         write "FRAM" chunk
       endif
       if(x_loc == 0 AND y_loc == 0) then
          write "DEFI saved_images 1 1 1" chunk
       else
          write "DEFI saved_images 1 1 1 x_loc y_loc" chunk
       write "<image>"
       write "SHOW 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
    if(loops > 1)write "DISC" and  "ENDL 0" chunks
    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 to convey it to anyone else.

8 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
- Josh M. Osborne, stripes@va.pubnix.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 04 Jul 1997