MNG-LC (Multiple-image Network Graphics—Low Complexity) Format Version 0.97

For list of authors, see Credits (Chapter 13).

Status of this Memo

This document is a specification by the PNG development group.

It has been approved by a vote of the group, and future technical changes will require formal approval by a vote of the group. It is the intent of the group to maintain backward compatibility if possible. We will, however, correct any technical deficiencies discovered in the course of developing "beta" implementations.

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

- mpng-list@ccrc.wustl.edu
- png-group@w3.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/.
```

In the case of any discrepancy between this extract and the full MNG specification, the full MNG specification shall take precedence.

Changes from sixty-ninth MNG draft (mng-0.96-19990717)

- Revised wording about frame count and layer count in MNG-VLC datastreams.
- Clarified that the BACK chunk components are always written as though for an RGBA PNG with 16-bit sample depth.
- Fixed some typos.

Abstract

This document presents the MNG-LC (Multiple-image Network Graphics, Low Complexity) format, which is a proper subset of the MNG (Multiple-image Network Graphics) format.

MNG is a multiple-image member of the PNG (Portable Network Graphics) format family that can contain animations (slide shows) or complex still frames, comprised of multiple PNG single-image datastreams.

The MNG-LC 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-LC datastream.

A MNG-LC 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 datastream.

This document includes examples that demonstrate various capabilities of MNG-LC including simple movies and composite frames.

Table of Contents

1 Introduction **5**

2 Terminology **7**

3 MNG Chunks **9**
3.1 Critical MNG control chunks . . . 9
3.1.1 MHDR MNG datastream header . . . 9
3.1.2 MEND End of MNG datastream . . . 11
3.1.3 LOOP, ENDL Define a loop . . . 12
3.2 Critical MNG image defining chunks . . . 12
3.2.1 DEFI Define an object . . . 12
3.2.2 PLTE and tRNS Global palette . . . 13
3.2.3 IHDR, PNG chunks, IEND . . . 14
3.2.4 JHDR, JNG chunks, IEND . . . 14
3.3 Critical MNG image displaying chunks . . . 15
3.3.1 BACK Background . . . 15
3.3.2 FRAM Frame definitions . . . 16
3.3.3 TERM Termination action . . . 22
3.4 SAVE and SEEK chunks . . . 23
3.5 Ancillary MNG chunks . . . 23
3.5.1 eXPI Export image . . . 23
3.5.2 pHYg Physical pixel size (global) . . . 24
3.6 Ancillary PNG chunks . . . 24

4 The JPEG Network Graphics (JNG) Format **25**

5 Chunk Copying Rules **25**

6 Minimum Requirements for MNG-LC-Compliant Viewers **25**
6.1 Required MNG chunk support . . . 26
6.2 Required PNG chunk support . . . 27
6.3 Optional JNG chunk support . . . 27

7 Recommendations for Encoders **28**
7.1 Use a common color space . . . 28
7.2 Use the right framing mode . . . 28
7.3 Immediate frame sync point . . . 29

8 Recommendations for Decoders **29**
8.1 Decoder handling of fatal errors . . . 29
8.2 Decoder handling of interlaced images . . . 29
8.3 Decoder handling of palettes . . . 29
8.4 Behavior of single-frame viewers . . . 29

8.5 Clipping 30

9 Miscellaneous Topics **30**
9.1 File name extension 30

10 Revision History **31**
10.1 Version 0.97 31
10.2 Version 0.96 31
10.3 Version 0.95 31

11 Security Considerations **31**

12 Appendix: Examples **33**
12.1 Example 1: A single image 33
12.2 Example 2: A very simple movie 33
12.3 Example 3: A simple slideshow 34
12.4 Examples 4-14: Omitted from MNG-LC. 34
12.5 Example 15: Converting a simple GIF animation 35
12.6 Example 16: Counting layers and frames 36
12.7 Example 17: Storing an icon library 37

13 Credits **38**

1 Introduction

This document presents a low-complexity version (MNG-LC, which is a proper subset) of the MNG (Multiple-image Network Graphics) format.

Note: This specification depends on the PNG (Portable Network Graphics) [PNG] and, for MNG-LC applications that are enhanced with JNG support, the JNG (JPEG Network Graphics) specifications. The PNG specification is available at the PNG web site,

`http://www.cdrom.com/pub/png/`

and the JNG (JPEG Network Graphics) specification and the full MNG specification are available at the MNG web site,

`http://www.cdrom.com/pub/mng/`

A MNG-LC datastream describes a sequence of zero or more single frames, each of which can be composed of zero or more embedded images.

The embedded images can be PNG or JNG datastreams. MNG-LC datastreams do not contain JNG (JPEG Network Graphics) datastreams, which are allowed in full MNG datastreams, but MNG-LC applications can be enhanced to recognize and process those as well.

A typical MNG-LC datastream consists of:

- The 8-byte MNG signature.
- The MHDR chunk.
- Frame definitions. A frame is one or more subframes, the last of which has a nonzero interframe delay, composited against whatever was already on the display,
- Subframe definitions. A subframe contains one or more layers, together with clipping information and an interframe delay.
- Layer definitions. A layer is:
  - An embedded image.
  - The background.
- The MEND chunk.

MNG is fundamentally declarative; it describes the elements that go into an individual frame. It is up to the decoder to work out an efficient way of making the screen match the desired composition whenever a nonzero interframe delay occurs. Simple decoders can handle it as if it were procedural, compositing the images into the frame buffer in the order that they appear, but efficient decoders might do something different, as long as the final appearance of the frame is the same.

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 "video/x-mng" can be used. Registration of the media type "video/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 0–3.

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.

The chunk copying rules for MNG employ the same mechanism as PNG, but with rules that are explained more fully in the full MNG specification.

Note that decoders are not required to follow any decoding models described in this specification nor to follow the instructions in this specification, as long as they produce results identical to those that could be produced by a decoder that did use this model and did follow the instructions.

Each chunk of the MNG datastream or of any embedded object 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-LC datastream that must be recognized and decoded by MNG-LC decoders. An independent JNG datastream must also be recognized and decoded by any MNG-LC decoder that has been enhanced to include JNG support. This kind of MNG-LC or enhanced MNG-LC datastream will contain only a single embedded image.

Because the embedded objects making up a MNG are normally 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.
- It is complementary to JPEG and does not attempt to replace JPEG for lossy storage of images (however, JNG-enhanced MNG-LC can accommodate JPEG-encoded images that are encoded in the PNG-like JNG format.

In addition:

- It provides animation with variable interframe delays.
- It allows composition of frames containing multiple images.
- Multiple-image GIF files (except for those using the "restore-to-previous" disposal method) can be losslessly converted to MNG-LC.

2 Terminology

See also the glossary in the PNG specification and the "terminology" section of the full MNG specification.

requirement levels
: The words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119, and the word "CAN" is equivalent to the word "MAY" as described therein. "NOT ALLOWED" and "NOT PERMITTED" describe conditions that "MUST NOT" occur. "ALLOWED" and "PERMITTED" describe conditions that "CAN" occur.

embedded image
: An image that appears in-line in a MNG datastream.

frame
: A layout of zero or more layers, with zero interframe delay time between them, followed by a specified nonzero delay time or by the MEND chunk, that is to be displayed as a still picture or as part of an animation. An animation would ideally appear to a perfect observer (with an inhumanly fast visual system) as a sequence of still pictures. A group of MNG images, with zero interframe delay time between them, and also followed by a zero interframe delay time, makes up a "subframe" (see below) instead of a "frame". The final subframe in the datastream completes a frame, regardless of its interframe delay.

  When the layers of a frame do not cover the entire area defined by the width and height fields from the MHDR chunk, the layers are composited over the previous frame to obtain the new frame.

frame origin
: The upper left corner of the output device (frame buffer, screen, window, page, etc.) where the pixels are to be displayed. This is the {0,0} position for the purpose of defining frame clipping boundaries, image locations, and image clipping boundaries. Note that in a windowing system, the frame origin might be moved offscreen, but the locations in DEFI chunks would still be measured from this offscreen origin.

interframe delay
: The amount of time a frame should be visible when an animation is played. In reality, it takes a nonzero amount of time to display a frame. No matter which moment is picked as the "start" of the

frame, the interframe delay measures the time to the "start" of the next frame. There is no interframe delay prior to the implicit background layer at the beginning of the animation nor after the final frame.

layer

A layer can be

- A visible embedded image, located with respect to the frame boundaries and clipped with respect to the subframe clipping boundaries and the image's own clipping boundaries.
- The background that is displayed before the first image in the entire datastream is displayed. Its contents can be defined by the application or by the BACK chunk.
- a solid rectangle filled with the background color and clipped to the subframe boundaries at the beginning of a subframe, when the framing mode is 3 or 4.

Note that a layer can be completely empty if the image is entirely outside the clipping boundaries.

A layer can be thought of as a transparent rectangle with the same dimensions as the frame, with an image composited into it, or it can be thought of as a rectangle having the same dimensions (possibly zero) and location as those of the object after it has been located and clipped.

An embedded visible PNG or JNG datastream generates a single layer, even though it might be interlaced or progressive.

MNG-LC

A low-complexity version of MNG in which only image 0 is permitted, loops can all be run once, JNG and Delta-PNG are not present, and certain MNG chunks are not present. Bit 0 of the MHDR simplicity profile must be 1, indicating that the profile is valid, and all other bits of the profile except possibly for bits 2 and 3 must be 0, indicating that "complex MNG features", JNG, and Delta-PNG are not present. Bits 2 and 3 can be 0 or 1, depending on whether "simple MNG features" and transparency are absent or present. MNG-LC is a proper subset of MNG.

object, object_id

An image. The `object_id` is an unsigned sixteen-bit number that serves as the identifier of a set of object attributes. In MNG-LC only object 0 is permitted.

object attributes

Properties of an object such as its existence, potential visibility, location, and clipping boundaries.

potentially visible image

A potentially visible image is

- a not-yet-defined object that is "marked" for on-the-fly display while the embedded image that defines it is being decoded or cloned, by setting its `do_not_show` flag to zero.

signal

An entity with a number that can arrive asynchronously at the decoder. More detailed semantics, like whether multiple signals of the same number (or even different numbers) can be queued, are beyond the scope of this specification.

subframe
A group of one or more layers that are to be displayed as a part of a still frame or as part of one frame of an animation. Subframes are defined in MNG for convenience in applying frame parameters to a subset of the layers making up a complete frame.

visible image
Actually drawn on a display. If an object is visible, a person looking at the display can see it.

3 MNG Chunks

This chapter describes chunks that can appear at the top level of a MNG datastream.

3.1 Critical MNG control chunks

This section describes critical MNG control chunks that MNG-LC-compliant decoders must recognize and process ("processing" a chunk sometimes can consist of simply recognizing it and ignoring it; some chunks have been declared to be critical only to prevent them from being relocated by MNG editors).

3.1.1 MHDR MNG datastream header

The MHDR chunk is always first in all MNG datastreams except for those that consist of a single PNG or JNG datastream with a PNG or JNG signature.

The MHDR chunk contains exactly 28 bytes:

```
Frame width:  4 bytes (unsigned integer).
Frame height: 4 bytes (unsigned integer).
Ticks per
 second:      4 bytes (unsigned integer).
Nominal layer
 count:       4 bytes (unsigned integer).
Nominal frame
 count:       4 bytes (unsigned integer).
Nominal play
 time:        4 bytes (unsigned integer).
Simplicity
 profile:     4 bytes:(unsigned integer).
                bit 0:
                       1: Presence or absence of certain
                          features is specified by the
                          remaining bits of the simplicity
                          profile.
                       (must be 1 in MNG-LC datastreams)
```

```
bit 1:
       0: Simple MNG features are absent.
       1: Simple MNG features are present.
bit 2:
       0: Complex MNG features are absent.
       (must be 0 in MNG-LC datastreams)
bit 3:
       0: transparency is absent or can be
          ignored.
       1: transparency is present and is
          essential (critical) for proper
          display of the images.
bit 4:
       0: JNG is absent.
       1: JNG is present.
       (must be 0 in MNG-LC datastreams)
bit 5:
       0: Delta-PNG is absent.
       (must be 0 in MNG-LC datastreams)
bits 6-15:
       Reserved for public expansion.  Must
       be zero in this version.
bits 16-30:
       Available for private or experimental
       expansion.  Undefined in this version
       and can be ignored.
bit 31:
       Must be zero.
```

Decoders can ignore the "informative" frame-count, layer-count, play-time, and simplicity-profile fields.

The `frame_width` and `frame_height` fields give the intended display size (measured in pixels) and provide the default clipping boundaries.

The `ticks_per_second` field gives the unit used by the FRAM chunk to specify interframe delay and timeout. It must be nonzero if the datastream contains an animation. When the datastream contains exactly one frame, this field should be set to zero. When this field is zero, the length of a tick is infinite, and decoders will ignore any attempt to define interframe delay, timeout, or any other variable that depends on the length of a tick. If the frames are intended to be displayed one at a time under user control, such as a slide show or a multi-page FAX, the tick length can be set to any positive number and a FRAM chunk can be used to set an infinite interframe delay and a zero timeout. Unless the user intervenes, viewers will only display the first frame in the datastream.

When `ticks_per_second` is nonzero, and there is no other information available about interframe delay, viewers should display animations at the rate of one frame per tick.

If the frame-count field contains a zero, the frame count is unspecified. If it is nonzero, it contains the number of frames that would be displayed, ignoring the TERM chunk. If the frame count is greater than $2^{31} - 1$, encoders should write $2^{31} - 1$, representing an infinite frame count.

If the layer-count field contains a zero, the layer count is unspecified. If it is nonzero, it contains the number of layers in the datastream, ignoring the TERM chunk. If the layer count is greater than $2^{31} - 1$, encoders should write $2^{31} - 1$, representing an infinite layer count.

If the nominal-play-time field contains a zero, the nominal play time is unspecified. Otherwise, it gives the play time, in ticks, when the file is displayed ignoring the TERM chunk. Authors who write this field should choose a value of "ticks_per_second" that will allow the nominal play time to be expressed in a four-bit integer. If the nominal play time is greater than $2^{31} - 1$ ticks, encoders should write $2^{31} - 1$, representing an infinite nominal play time.

When bit 0 of the simplicity profile is zero, the simplicity (or complexity) of the MNG datastream is unspecified, and *all* bits of the simplicity profile must be zero. The simplicity profile must be nonzero in MNG-LC datastreams.

If the simplicity profile is nonzero, it can be regarded as a 32-bit profile, with bit 0 (the least significant bit) being a "profile-validity" flag, bit 1 being a "simple MNG" flag, bit 2 being a "complex MNG" flag, bit 3 being a "transparency" flag, bit 4 being a "JNG" flag, and bit 5 being a "Delta-PNG" flag. The upper 15 bits (except for the most significant bit, which must be zero) are available for private test or experimental versions, and the remaining bits are reserved for future MNG versions, and must be zero in this version. If a bit is zero, the corresponding feature is guaranteed to be absent, and if a bit is one, the corresponding feature may be present in the MNG datastream.

When bit 1 is zero ("simple" MNG features are absent), the datastream does not contain the DEFI, FRAM, or global PLTE and tRNS chunks.

"Transparency is absent or can be ignored" means that the MNG or PNG tRNS chunk is not present and no PNG or JNG image has an alpha channel, or that if they are present they are not essential (or critical) for displaying the images.

A MNG-LC (i.e., a "low-complexity MNG") datastream must have a simplicity profile with bit 0 equal to 1 and all other bits except possibly for bits 1 and 3 ("simple MNG" MNG features and transparency) equal to zero. If bit 4 (JNG) is 1, the datastream is a "MNG-LC with JNG" datastream. MNG-LC decoders are allowed to reject such datastreams unless they have been enhanced with JNG capability.

Encoders that write a nonzero simplicity profile should endeavor to be accurate, so that decoders that process it will not unnecessarily reject datastreams. For example, the simplicity profile 31 indicates that JNG, critical transparency, and at least one "complex" MNG feature are all present, but Delta-PNG is not. This example would not qualify as a MNG-LC datastream because a "complex" MNG feature might be present. If the simplicity profile promises that certain features are absent, but they are actually present in the MNG datastream, the datastream is invalid.

3.1.2 MEND End of MNG datastream

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

3.1.3 LOOP, ENDL Define a loop

The LOOP chunk can be ignored by MNG-LC decoders, along with the ENDL chunk.

3.2 Critical MNG image defining chunks

3.2.1 DEFI Define an object

The DEFI chunk sets the default object attribute set (`object_id`, `potential_visibility`, `concrete_flag`, location, and clipping boundaries) for any subsequent images that are defined with IHDR-IEND, or JHDR-IEND datastreams.

Bit 1 of the MHDR simplicity profile can be used to promise that the DEFI chunk is not present.

The DEFI chunk contains 2, 3, 4, 12, or 28 bytes:

```
Object id:      2 bytes (unsigned integer) identifier
                  to be given to the objects that
                  follow the DEFI chunk.
                  This field must be zero in MNG-LC
                  files.

Do_not_show
 flag:          1 byte (unsigned integer)
                  0:  Make the objects potentially
                      visible.
                  1:  Do not make the objects potentially
                      visible.  This field can be omitted
                      if the concrete_flag, location, and
                      clipping boundary fields are also
                      omitted.

Concrete flag: 1 byte (unsigned integer)
                  0:  Make the objects "abstract" (image
                      cannot be the source for a
                      Delta-PNG)
                  1:  Make the objects "concrete" (object
                      can be the source for a Delta-PNG).
                  This field can be omitted if the
                  location and clipping boundary fields
                  are also omitted.
                  MNG-LC decoders can ignore this flag.

X_location:     4 bytes (signed integer).
                  The X_location and Y_location fields can
                  be omitted if the clipping boundaries
                  are also omitted.
```

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

Left_cb:       4 bytes (signed integer).  Left clipping
                 boundary.  The left_cb, right_cb,
                 top_cb, and bottom_cb fields can be
                 omitted as a group.

Right_cb:      4 bytes (signed integer).

Top_cb:        4 bytes (signed integer).

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

Negative values are permitted for the X and Y location and clipping boundaries. The positive directions are downward and rightward from the frame origin.

If no DEFI chunk precedes an object, the decoder must use the following default values:

```
          Object_id = 0
       Do_not_show = 0
    Concrete_flag = 0
       X_location = 0
       Y_location = 0
          Left_cb = 0
         Right_cb = frame_width
           Top_cb = 0
        Bottom_cb = frame_height
```

These default values are also used to fill any fields that were omitted from the DEFI chunk, when an object with the same `object_id` has not been previously defined or a DISC or SEEK chunk has caused it to be discarded.

The object attributes for all unfrozen objects, including object 0, become undefined when a SEEK chunk is encountered. Encoders must ensure that the first DEFI chunk for every object_id after any SEEK chunk does not omit any fields that were set to values other than these defaults in the previous DEFI chunk for that object_id, because simple decoders are allowed to ignore the SEEK chunk. Decoders that jump or skip to SEEK chunks must assume these defaults for any omitted fields in the first DEFI chunk in a segment for each object_id.

3.2.2 PLTE and tRNS Global palette

The PLTE chunk has the same format as a PNG PLTE chunk. It provides a global palette that is inherited by PNG datastreams that contain an empty PLTE chunk.

The tRNS chunk has the same format as a PNG tRNS chunk. It provides a global transparency array that is inherited along with the global palette by PNG datastreams that contain an empty PLTE chunk.

If a PNG datastream is present that does not contain an empty PLTE chunk, neither the global PLTE nor the global tRNS data is inherited by that datastream.

If the global PLTE chunk is not present, each indexed-color PNG in the datastream must supply its own PLTE (and tRNS, if it has transparency) chunks.

3.2.3 IHDR, PNG chunks, IEND

A PNG (Portable Network Graphics) datastream.

See the PNG specification [PNG] and the Extensions to the PNG Specification document [PNG-EXT] for the format of the PNG chunks.

Any chunks between IHDR and IEND are written and decoded according to the PNG specification.

If a global PLTE chunk appears in the top-level MNG datastream, the PNG datastream can have an empty PLTE chunk, to direct that the global PLTE and tRNS data be used. If an empty PLTE chunk is not present, the data is not inherited. MNG applications that recreate PNG files must write the global PLTE chunk rather than the empty one in the output PNG file, along with the global tRNS data if it is present. The global tRNS data can be subsequently overridden by a tRNS chunk in the PNG datastream. It is an error for the PNG datastream to contain an empty PLTE chunk when the global PLTE chunk is not present or has been nullified.

The PNG oFFs and pHYs chunks and any future chunks that attempt to set the pixel dimensions or the drawing location must be ignored by MNG viewers and simply copied (according to the copying rules) by MNG editors.

The PNG gIFg, gIFt, and gIFx chunks must be ignored by viewers and must be copied according to the copying rules by MNG editors.

If `do_not_show=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, or to display it after decoding is complete.

3.2.4 JHDR, JNG chunks, IEND

A JNG (JPEG Network Graphics) datastream.

See the JNG specification for the format of the JNG datastream.

Any chunks between JHDR and IEND are written and decoded according to the JNG specification.

The remaining discussion in the previous paragraph about PNG datastreams also applies to JNG datastreams.

MNG-LC applications are not expected to process JNG datastreams unless they have been enhanced with JNG capability.

3.3 Critical MNG image displaying chunks

3.3.1 BACK Background

The BACK chunk suggests a background color against which transparent, clipped, or less-than-full-frame images can be displayed.

```
Red_background:   2 bytes (unsigned integer).

Green_background: 2 bytes (unsigned integer).

Blue_background:  2 bytes (unsigned integer).

Mandatory
  background:     1 byte (unsigned integer).
                     0: Background color is advisory.
                        Applications can use it if they
                        choose to.
                     1: Background color is mandatory.
                        Applications must use it.
                     This byte can be omitted.  If so,
                     the background color is advisory.
```

Viewers are expected to composite the first layer in the MNG-LC datastream against a background layer that fills the entire frame. The BACK chunk provides a background that the viewer can use for this purpose. If it is not "mandatory" the viewer can choose another background if it wishes. If the BACK chunk is not present, or if it has been clipped to less than full frame, the viewer must provide its own background for the first subframe. Each subframe after the first must be composited over the layers that precede it.

Viewers are expected, however, to composite every subframe in the MNG datastream against a fresh copy of the background, when the framing mode given in the FRAM chunk is 3 or 4. The images and the background are both clipped to the frame boundaries given in the FRAM chunk. Anything outside these boundaries is inherited from the previous subframe.

The three BACK components are always written as though for an RGBA PNG with 16-bit sample depth. For example, a mid-level gray background could be specified with (0x9999, 0x9999, 0x9999). The background color is interpreted in the current color space as defined by any top-level gAMA, cHRM, iCCP, sRGB chunks that have appeared prior to the BACK chunk in the MNG datastream. If no such chunks appear, the color space is unknown.

When the BACK chunk appears between FRAM chunks, it applies to the upcoming frame, not to the current one. When the framing_mode is 3, it takes effect immediately prior to the next IHDR or JHDR chunk in the datastream.

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

The BACK chunk can be omitted. If a background is needed and the BACK chunk is omitted, then the viewer must supply its own background. For the purpose of counting layers, such a viewer-supplied background

layer is counted the same as a background supplied by the BACK chunk.

In practice, most applications that use MNG as part of a larger composition should ignore the BACK data if `mandatory_background=0` and the application already has its own background definition. This will frequently be the case in World Wide Web pages, to achieve nonrectangular transparent animations displayed against the background of the page.

3.3.2 FRAM Frame definitions

The FRAM chunk provides information that a decoder needs for generating layers, subframes, and frames. The FRAM parameters govern how the decoder is to behave when it encounters a FRAM chunk, or an embedded image. The FRAM chunk also delimits subframes.

Bit 1 of the MHDR simplicity profile can be used to promise that the FRAM chunk is not present.

An empty FRAM chunk is just a subframe delimiter. A nonempty one is a subframe delimiter, and it also changes FRAM parameters, either for the upcoming subframe or until reset. When the FRAM chunk is not empty, it contains a framing-mode byte, an optional name string, a zero-byte separator, plus four 1-byte fields plus a variable number of optional fields.

```
Framing mode:  1 byte.

                0:  Don't change framing mode.

                1:  No background layer is inserted.
                    The interframe delay occurs before
                    each image (except the very first
                    in the datastream) is displayed.

                2:  No background layer is inserted.
                    The interframe delay occurs before
                    each "FRAM" chunk.  No delay occurs
                    before or between the images.

                3:  The background layer is displayed
                    before each image is displayed.
                    The interframe delay occurs before
                    displaying the background, and no
                    delay occurs between the background
                    and the image.

                4:  The background layer is inserted
                    after each "FRAM" chunk.  The
                    interframe delay occurs before
                    the background is displayed, and no
                    delay occurs before or between the
                    images.
```

```
Subframe name: 0 or more bytes (Latin-1 Text).  Can be
                 omitted; if so, the subframe is nameless.

Separator:     1 byte:  (null).  Must be omitted if all
                 remaining fields are also omitted.

Change interframe
    delay:     1 byte.

                 0:  No.
                 1:  Yes, for the next subframe only.
                 2:  Yes, also reset default.

               This field and the next three must be
               omitted as a group if no frame parameters
               other than the framing mode are changed.

Change timeout and
  termination: 1 byte
                 0:  No.
                 1:  Deterministic, for the next subframe
                     only.
                 2:  Deterministic, also reset default.
                 3:  Decoder-discretion, for the next
                     subframe only.
                 4:  Decoder-discretion, also reset
                     default.
                 5:  User-discretion, for the next subframe
                     only.
                 6:  User-discretion, also reset default.
                 7:  External-signal, for the next subframe
                     only.
                 8:  External-signal, also reset default.

Change subframe clipping
 boundaries:   1 byte.
                 0:  No.
                 1:  Yes, for the next subframe only.
                 2:  Yes, also reset default.

Change sync id
  list:        1 byte.
                 0:  No.
                 1:  Yes, for the next subframe only.
                 2:  Yes, also reset default list.

Interframe
  delay:       4 bytes (unsigned integer).  Must
                 be omitted if change_interframe_delay=0.
```

```
                  The range is [0..2^31-1] ticks.

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

Subframe boundary
  delta type:  1 byte (unsigned integer).
                  0: Subframe clipping boundary values are
                     given directly.
                  1: Subframe clipping boundaries
                     are determined by adding the FRAM
                     data to their previous values.
               This and the following four
               fields must be omitted if
               change_frame_clipping_boundaries=0.

Left_fb or delta
  left_fb:     4 bytes (signed integer).

Right_fb or delta
  right_fb:    4 bytes (signed integer).

Top_fb or delta
  top_fb:      4 bytes (signed integer).

Bottom_fb or delta
  bottom_fb:   4 bytes (signed integer).

Sync id:       4 bytes (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 subframe that is about to be defined, not the one that is being terminated by the FRAM chunk.

Framing modes:

The framing mode provides information to the decoder that it uses whenever it is about to display an image, and when it is processing the *next* FRAM chunk.

The following events trigger the display of an image:

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

- Also, decoding the next FRAM chunk, when the framing mode is 3 or 4 and none of the above were found between the FRAM chunks.

Any of these events generates a display update, even if the visible image is outside the clipping boundaries and no pixels are actually changed.

When a decoder is ready to perform a display update, it must check the framing mode, to decide whether it should restore the background (framing modes 3 and 4) or not (framing modes 1 and 2), and whether it needs to wait for the interframe delay to elapse before continuing (framing modes 1 and 3) or not (framing modes 2 and 4).

When the interframe delay is zero, viewers are not required actually to update the display but can continue to process the remainder of the frame and composite the next image over the existing frame before displaying anything. The final result must appear the same as if each image had been displayed in turn with no delay.

Regardless of the framing mode, encoders must insert a background layer, with a zero delay, ahead of the first image layer in the datastream, even when the BACK chunk is not present or has been clipped to less than full-frame. This layer must be included in the layer count but not in the frame count.

Framing mode 1.
When `framing_mode=1`, the decoder must wait until the interframe delay has elapsed before displaying each image. The background layer is inserted, with no interframe delay, before the first image in the datastream. If the interframe delay is nonzero, the first frame in the datastream consists of two layers and the rest consist of a single layer.

Framing mode 2.
Framing mode 2 is the same as framing mode 1, except that the interframe delay occurs between subframes rather than between images.

In the usual case, the interframe delay is nonzero, so each subframe is a composite frame. When `framing_mode=2`, viewers are expected to display all of the images in a frame at once, if possible, or as fast as can be managed, without clearing the display or restoring the background. The next FRAM chunk delimits the subframe. A subframe also ends when a SEEK chunk or the MEND chunk appears, and the final frame ends when the MEND chunk appears.

Framing mode 3.
When `framing_mode=3`, a background layer is displayed before each image layer is displayed, without a delay between the background and the image. The decoder must wait until the interframe delay has elapsed before displaying the background. If the interframe delay is nonzero, each frame consists of two layers. A subframe boundary occurs after each image appears. Otherwise, `framing_mode=3` is identical to `framing_mode=1`.

Framing mode 4.
When `framing_mode=4`, the background layer is displayed before each frame, i.e., after each FRAM chunk, with no interframe delay before each image. The decoder must wait until the interframe delay has elapsed before displaying the background layer. Otherwise, `framing_mode=4` is identical to `framing_mode=2`.

The subframe name must conform to the same formatting rules as those for a PNG 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 (unless the subframe name is omitted) and no more than 79 characters in the keyword. Keywords are case-sensitive. There is no null byte within the keyword. Applications can use this field for such purposes as constructing an external list of subframes in the datastream. The subframe name only applies to the upcoming subframe; subsequent subframes are unnamed unless they also have their own `frame_name` field. It is recommended that the same name not appear in any other FRAM chunk or in any SEEK or eXPI chunk. Subframe names should not begin with the case-insensitive strings "CLOCK(", "FRAME(", or "FRAMES(", which are reserved for use in URI queries and fragments, as explained in the full MNG specification.

The interframe delay value is the desired minimum time to elapse from the beginning of displaying one frame until the beginning of displaying the next frame. When the interframe delay is nonzero, which will probably be the usual case, subframes are frames. When it is zero, a frame consists of any number of consecutive subframes, until a nonzero delay subframe is encountered and completed. Decoders are not obligated to display such subframes individually; they can composite them offscreen and only display the complete frame.

There is no interframe delay before the first layer (the implicit background layer) in the datastream nor after the final frame, regardless of the framing mode.

The timeout field can be a number or <infinity>. Infinity can be represented by 0x7fffffff.

The termination condition given in the `change_timeout_and_termination` field specifies how much longer, after the normal interframe delay has elapsed, the frame will endure, and whether the interframe delay can be shortened. It can take the following values:

deterministic
: The frame endures no longer than the normal interframe delay. Even though this is the default, a streaming encoder talking to a real-time decoder might write a FRAM with a termination condition of "deterministic" to force the display to be updated while the encoder decides its next move.

decoder-discretion
: If the interframe delay is nonzero, the decoder can lengthen the duration of the frame, but by no more than the timeout. A streaming decoder could take the opportunity to wait for its input buffer to fill to a comfortable level. When the timeout is zero, the decoder can shorten the duration of the frame to less than the interframe delay.

user-discretion
: If the interframe delay is nonzero, after the interframe delay has expired, the decoder should wait for permission from the user (e.g., via a keypress) before proceeding, but must wait no longer than the timeout. If the decoder cannot interact with the user, this condition degenerates into "decoder-discretion". When the timeout is zero, the user can shorten the duration of the frame to less than the interframe delay.

external-signal
: If the interframe delay is nonzero, after the interframe delay has expired, the decoder should wait for the arrival of a signal whose number matches a `sync_id`, but must wait no longer than the

timeout. When the timeout is zero, the arrival of a signal can shorten the duration of the frame to less than the interframe delay.

The sync_id list can be omitted if the termination condition is not "external-signal".

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
Subframe name             = <empty string>
Interframe delay          = 1
Left subframe boundary    = 0
Right subframe boundary   = frame width
Top subframe boundary     = 0
Bottom subframe boundary  = frame height
Termination               = deterministic
Timeout                   = 0x7fffffff (infinite)
Sync id                   = <empty list>
```

The subframe boundaries from the FRAM chunk are only used for clipping, not for placement. The DEFI chunk can be used to specify the placement of each image within the layer. The DEFI chunk can be used to specify clipping boundaries for each image. Even when the left and top subframe boundaries are nonzero, the image locations are measured with respect to the {0,0} position in the display area.

If the layers are transparent or do not cover the entire area defined by the subframe clipping boundaries, they are composited against whatever already occupies the area, when the framing mode is 1 or 2. When the framing mode is 3 or 4, 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 does not define a mandatory background. The images, as well as the background, are clipped to the subframe clipping boundaries. Any pixels outside the subframe clipping boundaries remain unchanged from the previous subframe.

The interframe_delay field gives the duration of display, which is the minimum time that must elapse from the beginning of displaying one subframe until the beginning of displaying the next. It is measured in "ticks" using the tick length determined from ticks_per_second defined in the MHDR chunk. When the interframe delay is zero, it indicates that all subframes are to be combined into a single frame, until a nonzero interframe delay is specified or the MEND chunk is reached.

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 interframe_delay and the 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 subframe that follows the FRAM chunk.

The identifier sync_id=0 is reserved to represent synchronization with a user input from a keyboard or pointing device. The sync_id values 1–255 are reserved to represent the corresponding ASCII letter, received from the keyboard (or a simulated keyboard), and values 256–1023 are reserved for future definition by this specification. If multiple channels (not defined in this specification) are not present, viewers can ignore other values appearing in the sync_id list.

3.3.3 TERM Termination action

The TERM chunk suggests how the end of the MNG datastream should be handled, when a MEND chunk is found. It contains either a single byte or ten bytes:

```
Termination
  action:      1 byte (unsigned integer)
                 0: Show the last frame indefinitely.
                 1: Cease displaying anything.
                 2: Show the first frame after the TERM
                    chunk.
                 3: Repeat the animation starting
                    immediately after the TERM chunk.

Action after
  iterations: 1 byte
                 0: Show the last frame indefinitely after
                    iteration_max iterations have been
                    done.
                 1: Cease displaying anything.
                 2: Show the first frame after the TERM
                    chunk.

               This and the remaining fields must be
               present if termination_action is 3, and
               must be omitted otherwise.

Delay:         4 bytes (unsigned integer).  Delay, in
               ticks, before repeating the animation.

Iteration max: 4 bytes (unsigned integer).  Maximum
               number of times to repeat the animation.
```

The TERM chunk, if present, must appear either immediately after the MHDR chunk or immediately prior to a SEEK chunk. Only one TERM chunk is permitted in a MNG datastream.

Simple viewers and single-frame viewers can ignore the TERM chunk. It has been made critical only so MNG editors will not inadvertently relocate it.

3.4 SAVE and SEEK chunks

Simple decoders that only read MNG datastreams sequentially can safely ignore the SAVE and SEEK chunks.

3.5 Ancillary MNG chunks

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

3.5.1 eXPI Export image

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

The chunk contains an object identifier (snapshot id) and a name:

```
Snapshot id:   2 bytes (unsigned integer).
               (must be zero in MNG-LC
               datastreams)
Snapshot name: 1-79 bytes (Latin-1 text).
```

When the snapshot_id is zero, the snapshot is the first instance of an embedded image following the eXPI chunk.

Note that the `snapshot_name` is associated with the snapshot, not with the `snapshot_id` nor its future contents; changing the image identified by `snapshot_id` will not affect the snapshot. The `snapshot_name` means nothing inside the scope of the MNG-LC specification. If two eXPI chunks use the same name, it is the outside world's problem (and the outside world's prerogative to regard it as an error). It is recommended, however, that the `snapshot_name` not be the same as that appearing in any other eXPI chunk or in any FRAM chunk. 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, to allow applications to extract images by their `snapshot_id`.

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 images, but they would normally have smaller dimensions and fewer colors than the MNG frames. They can be defined with the potential visibility field set to "invisible" if they are not intended to be shown as a part of the regular display.

The `snapshot_name` string must follow the format of a tEXt keyword: It must consist only of printable Latin-1 characters and must not have leading or trailing blanks, but can have single embedded blanks. There

must be at least one and no more than 79 characters in the keyword. Keywords are case-sensitive. There is no null byte terminator within the `snapshot_name` string, nor is there a separate null byte terminator. Snapshot names should not begin with the case-insensitive strings "CLOCK(", "FRAME(", or "FRAMES(" which are reserved for use in URI queries and fragments.

Multiple instances of the eXPI chunk are permitted in a MNG datastream, and they need not have different values of `snapshot_id`.

3.5.2 pHYg Physical pixel size (global)

The MNG pHYg chunk is identical in syntax to the PNG pHYs chunk. It applies to complete full-frame MNG layers and not to the individual images within them.

Conceptually, a MNG viewer that processes the pHYg chunk will first composite each image into a full-frame layer, then apply the pHYg scaling to the layer, and finally composite the scaled layer into the subframe. MNG datastreams can include both the pHYs chunk (either at the MNG top level or within the PNG and JNG datastreams) and the pHYg chunk (only at the MNG top level), to ensure that the images are properly displayed either when displayed by a MNG viewer or when extracted into a series of individual PNG or JNG datastreams and then displayed by a PNG or JNG application. The pHYs and pHYg chunks would normally contain the same values, but this is not necessary.

The MNG top-level pHYg chunk can be nullified by a subsequent empty pHYg chunk appearing in the MNG top level.

3.6 Ancillary PNG chunks

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

- iTXt, tEXt, zTXt
- tIME Same format as in PNG. Can appear at most once in the prologue segment (before the first SEEK chunk), and at most once per segment (between two consecutive SEEK chunks). In the prologue it indicates the last time any part of the MNG was modified. In a regular segment (between SEEK chunks), it indicates the last time that segment was modified.

  A MNG editor that writes PNG datastreams should not include the top-level iTXt, tEXt, tIME, and zTXt chunks in the generated PNG datastreams.
- cHRM, gAMA, iCCP, sRGB, bKGD, sBIT, pHYs

  These 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. Any of these chunks can be nullified by the appearance of a subsequent empty chunk with the same chunk name. Such empty chunks are not legal PNG or JNG chunks and must only appear in the MNG top level.

In the MNG top level, all of these chunks are written as though for 16-bit RGBA PNG datastreams. Decoders are responsible for reformatting the chunk data to suit the actual bit depth and color type of the datastream that inherits them.

A MNG editor that writes PNG or JNG datastreams is expected to include the top-level cHRM, gAMA, iCCP, and sRGB chunks in the generated PNG or JNG datastreams, if the embedded image does not contain its own chunks that define the color space. When it writes the sRGB chunk, it should write the gAMA chunk (and perhaps the cHRM chunk), in accordance with the PNG specification, even though no gAMA or cHRM chunk is present in the MNG datastream. It is also expected to write the pHYs chunk and the reformatted top-level bKGD chunk in the generated PNG or JNG datastreams, and the reformatted sBIT chunk only in generated PNG datastreams, when the datastream does not have its own bKGD, pHYs, or sBIT chunks.

4 The JPEG Network Graphics (JNG) Format

JNG (JPEG Network Graphics) is the lossy sub-format for MNG objects. It is described in the full MNG specification and is also available as a separate extract from the full MNG specification. Both documents are available at the MNG home page,

`http://www.cdrom.com/pub/mng/`

MNG-LC applications can choose to support JNG or not. Those that do not can check bit 4 (JNG is present/ absent) of the MHDR simplicity profile to decide whether they can process the datastream.

5 Chunk Copying Rules

The chunk copying rules for MNG are similar to those in PNG. Authors of MNG editing applications should consult the full MNG specification for details.

6 Minimum Requirements for MNG-LC-Compliant Viewers

This section specifies the minimum level of support that is expected of MNG-LC-compliant decoders, and provides recomendations for viewers that will support slightly more than the minimum requirements. All critical chunks must be recognized, but some of them can be ignored after they have been read and recognized. Ancillary chunks can be ignored, and do not even have to be recognized.

Applications that provide less than minimal MNG support should check the MHDR “simplicity profile” for the presence of features that they are unable to support. A specific subset, in which “complex MNG features” and JNG are absent, is called *“MNG-LC”*. In MNG-LC datastreams, bit 0 of the simplicity profile must be 1 and bits 2 and 4 must be 0. Another subset is called *“MNG-VLC”*. In MNG-VLC datastreams, “simple MNG features” are also absent, and bit 1 must therefore also be 0.

Subsets are useable when the set of MNG datastreams to be processed is known to be (or is very likely to be) limited to the feature set in MNG-LC. Limiting the feature set in a widely-deployed WWW browser to anything less than MNG with 8-bit JNG support would be highly inappropriate.

Some subsets of MNG support are listed in the following table, more or less in increasing order of complexity.

Level of support	MHDR Profile bits 31	30–6	5	4	3	2	1	0	Profile value
MNG–VLC without transparency	0	0	0	0	0	0	0	1	1
MNG–VLC	0	0	0	0	1	0	0	1	9
MNG–VLC with JNG	0	0	0	1	1	0	0	1	25
MNG–LC	0	0	0	0	1	0	1	1	11
MNG–LC with JNG	0	0	0	1	1	0	1	1	19

One reasonable path for an application developer to follow might be to develop and test the application at each of the following levels of support in turn:

1. MNG-VLC,
2. MNG-LC,
3. MNG-LC with JNG,
4. MNG (according to the full MNG specification).

An equally reasonable development path might be

1. MNG-VLC with JNG,
2. MNG-LC with JNG,
3. MNG (according to the full MNG specification).

On the other hand, a developer working on an application for storing multi-page fax documents might have no need for more than "MNG-VLC without transparency".

6.1 Required MNG chunk support

MHDR

The `ticks_per_second` must be supported by animation viewers. The simplicity profile, frame count, layer count, and nominal play time can be ignored. Decoders that provide less than minimal support can use the simplicity profile to identify datastreams that they are incapable of processing.

MEND

The MEND chunk must be recognized but does not require any processing other than completing the last frame.

Global PLTE and tRNS
Must be fully supported. Bit 1 of the simplicity profile can be used to promise that these chunks are not present.

DEFI, BACK
Must be fully supported.

FRAM
The `framing_mode` and clipping parameters must be supported. The `interframe_delay` must be supported except by single-frame viewers. The `sync_id` and `timeout` data can be ignored.

LOOP, ENDL, SAVE, SEEK, TERM
Must be recognized but can be ignored.

6.2 Required PNG chunk support

IHDR, PLTE, IDAT, IEND
All PNG critical chunks must be fully supported. All values of `color_type`, `bit_depth`, `compression_method`, `filter_method` and `interlace_method` must be supported (interlacing, as in PNG, need not necessarily be displayed on-the-fly; the image can be displayed after it is fully decoded). The alpha-channel must be supported, at least to the degree that fully opaque pixels are opaque and fully transparent ones are transparent. It is recommended that alpha be fully supported. Alpha can be ignored if bit 3 of the simplicity profile is 0.

tRNS
The PNG tRNS chunk, although it is an ancillary chunk, must be supported in MNG-compliant viewers, at least to the degree that fully opaque pixels are opaque and fully transparent ones are transparent. It is recommended that alpha data from the tRNS chunk be fully supported in the same manner as alpha data from an RGBA image or a JNG with an alpha channel contained in IDAT chunks. The tRNS chunk can be ignored if bit 3 of the simplicity profile is 0.

Other PNG ancillary chunks
Ancillary chunks other than PNG tRNS can be ignored, and do not even have to be recognized.

Color management
It is highly recommended that decoders support at least the gAMA chunk to allow platform-independent color rendering.

6.3 Optional JNG chunk support

Bit 4 of the simplicity profile can be used to promise that JNG chunks are not present. Viewers that choose not to support JNG can check this bit before deciding to proceed. MNG-LC decoders are not required to support JNG.

JHDR, JDAT, IDAT, JSEP, IEND
: All JNG critical chunks must be fully supported. All values of `color_type`, `bit_depth`, `compression_method`, `filter_method` and `interlace_method` must be supported (interlacing, as in PNG, need not necessarily be displayed on-the-fly; the image can be displayed after it is fully decoded). The alpha-channel must be supported, at least to the degree that fully opaque pixels are opaque and fully transparent ones are transparent. It is recommended that alpha be fully supported.

JNG ancillary chunks
: All JNG ancillary chunks can be ignored, and do not even have to be recognized.

JNG image sample depth
: Only `image_sample_depth=8` must be supported. The JSEP chunk must be recognized and must be used by minimal decoders to select the eight-bit version of the image, when both eight-bit and twelve-bit versions are present, as indicated by `image_sample_depth=20` in the JHDR chunk. When `image_sample_depth=12`, minimal decoders are not obligated to display anything. Such decoders can choose to display nothing or an empty rectangle of the width and height specified in the JHDR chunk.

7 Recommendations for Encoders

The following recommendations do not form a part of the specification.

7.1 Use a common color space

It is a good idea to use a single color space for all of the layers in an animation, where speed and fluidity are more important than exact color rendition. This is best accomplished by defining a single color space at the top level of MNG, using and either an sRGB chunk or the gAMA and cHRM chunks and perhaps the iCCP chunk, and removing any color space chunks from the individual images after converting them to the common color space.

When the encoder converts all images to a single color space before putting them in the MNG datastream, this will allow decoders to improve the speed and consistency of the display.

For single-frame MNG datastreams, however, where decoding speed is less important and exact color rendition might be more important, it is best to leave the images in their original color space, as recommended in the PNG specification, to avoid any loss of data due to conversion, and to retain the individual color space chunks if the images have different color spaces.

7.2 Use the right framing mode

Always use framing mode 1 or 2 when all of the images are opaque. This avoids unnecessary screen clearing, which can cause flickering.

7.3 Immediate frame sync point

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 `interframe_delay=0`, `timeout`, and `sync_id`, and the second establishing the synchronization point:

```
FRAM 2 0 1 1 0 1 0000 timeout sync_id
FRAM 0 name
```

8 Recommendations for Decoders

8.1 Decoder handling of fatal errors

When a fatal error is encountered, such as a bad CRC or an unknown critical MNG chunk, minimal viewers should simply abandon the MNG datastream.

8.2 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.

8.3 Decoder handling of palettes

When a PLTE chunk is received, it only affects the display of the PNG datastream that includes or inherits it. Decoders must take care that it does not retroactively affect anything that has already been decoded.

If a frame contains two or more images, the PLTE chunk in one image does not affect the display of 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.

8.4 Behavior of single-frame viewers

Viewers that can only display a single frame must display the first frame that they encounter.

8.5 Clipping

MNG-LC provides three types of clipping, in addition to any clipping that might be required due to the physical limitations of the display device.

Frame width and frame height
: The `frame_width` and `frame_height` are defined in the MHDR chunk and cannot be changed by any other MNG chunk.

  Decoders can use these parameters to establish the size of a window in which to display the MNG frames. When the `frame_width` or `frame_height` exceeds the physical dimensions of the display hardware, the contents of the area outside those dimensions is undefined. If a viewer chooses, it can create "scroll bars" or the like, to enable persons to pan and scroll to the offscreen portion of the frame. If this is done, then the viewer is responsible for maintaining and updating the offscreen portion of the frame.

  In the case of a MNG datastream that consists of a PNG or JNG datastream, with the PNG or JNG signature, the `frame_width` and `frame_height` are defined by the `width` and `height` fields of the IHDR (or JHDR) chunk.

Subframe clipping boundaries
: The subframe clipping boundaries are optionally defined in the FRAM chunk, and cannot be changed within a subframe. When the framing mode is 3 or 4, viewers must, prior to displaying each subframe, clear the area within the subframe clipping boundaries to the background color, thus creating a separate layer at the beginning of each subframe. Viewers must not change any pixels outside the subframe boundaries; encoders must be able to rely on the fact that the part of the display that is outside the subframe clipping boundaries (but inside the area defined by `frame_width` and `frame_height`) will remain on the display from frame to frame without being explicitly redisplayed.

Image clipping boundaries
: The image clipping boundaries are defined in the DEFI chunk. They are associated with individual objects, not with the subframes, and they can be changed within a subframe. They are useful for exposing only a portion of an image in a frame.

9 Miscellaneous Topics

9.1 File name extension

On systems where file names customarily include an extension signifying file type, the extension `.mng` is recommended for MNG (including MNG-LC) files. Lowercase `.mng` is preferred if file names are case-sensitive. The extension `.jng` is recommended for JNG files.

10 Revision History

10.1 Version 0.97

Released 28 February 2000.

Minor editorial changes only.

10.2 Version 0.96

Released 18 July 1999.

The changes that are not simple editorial changes were approved by votes of the PNG Development group that closed 16 July 1999 (pHYg and change to treatment of the pHYs chunk), 14 July 1999 (global bKGD and sBIT) and 25 June 1999 (change to LOOP chunk and treatment of the DEFI chunk and nonviewable objects).

- An object "comes into existence" when it is named in a DEFI chunk instead of later, when the corresponding embedded image is received.
- The special treatment of the object attributes set for object 0 was eliminated.
- If fields are omitted from the DEFI chunk, values are inherited from a previous DEFI chunk, if one was present. In MNG-0.95, such fields assumed specified default values. In this version, the default values are only used if no prior DEFI chunk with the same object_id was present or if the prior DEFI chunk has been discarded.
- Revised Example 14.
- Started a Revision History section.
- Added the pHYg chunk and changed the meaning of the global pHYs chunk.
- Added the global bKGD and sBIT chunks.

10.3 Version 0.95

Initial public release, approved by the PNG Development Group on 11 May 1999.

11 Security Considerations

Security considerations are addressed in the PNG specification.

Some people may experience epileptic seizures when they are exposed to certain kinds of flashing lights or patterns that are common in everyday life. This can happen even if the person has never had any epileptic

seizures. All graphics software and file formats that support animation and/or color cycling make it possible to encode effects that may induce an epileptic seizure in these individuals. It is the responsibility of authors and software publishers to issue appropriate warnings to the public in general and to animation creators in particular.

No known additional security concerns are raised by this format.

12 Appendix: Examples

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

12.1 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.
sRGB 2
IHDR 720 468 8 0 0 0 0  # Width and Height, etc.
IDAT ...
IEND
```

If `file.png` contains an sRGB chunk and also gAMA and cHRM chunks that are recommended in the PNG specification for "fallback" purposes, you can remove those gAMA and cHRM chunks from `file.mng` because any MNG viewer that processes the gAMA chunk is also required to recognize and process the sRGB chunk, so those chunks will always be ignored. Any MNG editor that converts the MNG file back to a PNG file is supposed to insert the recommended gAMA and cHRM chunks.

12.2 Example 2: A 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:

```
\212 M N G \r \n ^z \n  # MNG signature.
MHDR 256 300   # Width and height.
     1         # 1 tick per second.
     5 4 4     # Layers, frames, play time
     1         # MNG-VLC simplicity
TERM 3 0 120 10   # When done, repeat animation 10 times.
IHDR ...  IDAT ...  IEND # Four PNG datastreams
IHDR ...  IDAT ...  IEND # are read and displayed.
IHDR ...  IDAT ...  IEND
IHDR ...  IDAT ...  IEND
MEND
```

12.3 Example 3: A simple slideshow

```
\212 M N G \r \n ^z \n  # MNG signature.
MHDR 720 468 1 # Width and height, 1 tick per second.
     6 5 5     # Layers, frames, play time.
     1         # Simplicity profile (MNG-VLC)
FRAM 1 0 2 2 0  2 1 600 0 # Set interframe_delay to 1,
  # timeout to 600 sec, and sync_id list to {0}.
SAVE
SEEK "Briefing to the Workforce"
IHDR ...  IDAT ...  IEND  # DEFI 0, visible, abstract
SEEK "Outline"            # is implied.
IHDR ...  IDAT ...  IEND
SEEK "Our Vision"       IHDR ...  IDAT ...  IEND
SEEK "Our Mission"      IHDR ...  IDAT ...  IEND
SEEK "Downsizing Plans" IHDR ...  IDAT ...  IEND
MEND
```

12.4 Examples 4-14: Omitted from MNG-LC.

These examples in the full MNG specification use features that are not available in MNG-LC.

12.5 Example 15: Converting a simple GIF animation

Outline of a program to convert simple GIF animations that do not use the "restore-to-previous" disposal method to "simple" MNG (or "MNG-LC") format:

```
begin
   write "MHDR" chunk
   Interframe_delay := 0; Previous_mode := 1
   Framing_mode := 1
   if(loops>1) "write TERM 3 0 0 loops"
   write "mandatory BACK" chunk
   for subimage in gif89a file do
      if(interframe_delay != gif_duration) then
         interframe_delay := gif_duration
         write "FRAM 0 0 2 2 0 2 0 interframe_delay 0"
      endif
      if(X_loc != 0 OR Y_loc != 0) then
         write "DEFI 0 0 0 X_loc Y_loc" chunk
      endif
      write "<image>"
      if (gif_disposal_method < 1) then
         /* (none or keep) */
         Framing_mode := 1
      else if (gif_disposal_method == 2) then
         /* (restore background) */
         write "FRAM 4 0 1 0 1 0 0 L R T B"
         Previous_mode := 4; Framing_mode := 1
      else if (gif_disposal_method == 3) then
         /* (restore previous) */
         error ("can't do gif_disposal method = previous.")
      endif
      if(Framing_mode != Previous_mode) then
         write "FRAM Framing_mode" chunk
         Previous_mode := Framing_mode
      endif
   end
   write "MEND" chunk
end
```

Where "<image>" represents a PNG datastream containing a GIF frame that has been converted to PNG format.

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

12.6 Example 16: Counting layers and frames

This demonstrates the determination of the layer-count and frame-count that should be written in the MHDR chunk. For framing_modes 1 and 2, the FRAM chunks themselves don't generate layers. For framing_modes 3 and 4, they do generate layers ("B" for background), and also generate frames if there is no embedded image with which to combine the background layer. Note that every framing_mode creates a "B" layer at the beginning.

Given the following chunk stream:

```
MHDR sRGB Fn F I I I F F I I I F F I I I MEND
```

in which

```
Fn represents a FRAM chunk with framing_mode n
F represents an empty FRAM chunk;
I represents an embedded image
```

This table shows the layer count and frame count for each of the four possible values of framing-mode:

Framing mode	Layer count		Frame count	
1	B,I,I,I, I,I,I, I,I,I	= 10	BI,I,I, I,I,I, I,I,I	= 9
2	B,I,I,I, I,I,I, I,I,I	= 10	BIII,III,III	= 3
3	3*(B, B,I, B,I, B,I)	= 21	3*(B,BI,BI,BI)	= 12
4	3*(B,B,I,I,I)	= 15	B,BIII,B,BIII,B,BIII	= 6

12.7 Example 17: Storing an icon library

Here is an example of storing a library of icons in a MNG-LC datastream. All of the icons use the same palette, transparency, and colorspace, so these are put in global chunks at the beginning. Empty PLTE chunks in the embedded images are used to import the global palette and transparency data.

```
MHDR 96 96 1 6 5 5 11  # Profile 11 is MNG-LC
sRGB 2                 # Global sRGB
PLTE ...               # Global PLTE
tRNS 0                 # Global tRNS
eXPI 0 "thumbnail"
IHDR 32 32 ... PLTE IDAT ... IEND
eXPI 0 "left arrow"
IHDR 96 96 ... PLTE IDAT ... IEND
eXPI 0 "right arrow"
IHDR 96 96 ... PLTE IDAT ... IEND
eXPI 0 "up arrow"
IHDR 96 96 ... PLTE IDAT ... IEND
eXPI 0 "down arrow"
IHDR 96 96 ... PLTE IDAT ... IEND
MEND
```

13 Credits

Editor

- Glenn Randers-Pehrson randeg @ alum.rpi.edu

Contributors

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, oliver @ fromme.com
- Jean-loup Gailly, jloup @ gzip.org
- Chris Herborth, chrish @ qnx.com
- Alex Jakulin, jakulin @ acm.org
- Gerard Juyn, gjuyn @ xs4all.nl
- Neal Kettler, neal @ westwood.com
- 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 @ acm.org

- Josh M. Osborne, stripes @ va.pubnix.com
- Keith S. Pickens, ksp @ rice.edu
- Glenn Randers-Pehrson, randeg @ alum.rpi.edu
- Nancy M. Randers-Pehrson, randeg @ alum.rpi.edu
- Greg Roelofs, newt @ pobox.com
- Willem van Schaik, willem @ schaik.com
- Guy Schalnat, gschal @ infinet.com
- Paul Schmidt, pschmidt @ photodex.com
- Smarry Smarasderagd, smar @ reptiles.org
- Thomas R. Tanner, ttehtann @ argonet.co.uk
- Guido Vollbeding, guivol @ esc.de
- Tim Wegner, twegner @ phoenix.net
- Alaric B. Williams, alaric @ alaric-williams.com

Trademarks

- GIF is a service mark of CompuServe Incorporated.

Document source

This document was built from the file `mng-master-20000228` on 29 February 2000.

Copyright Notice

Copyright © 1998, 1999, 2000 by: Glenn Randers-Pehrson

This specification is being provided by the copyright holder under the following license. By obtaining, using and/or copying this specification, you agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute this specification for any purpose and without fee or royalty is hereby granted, provided that the full text of this **NOTICE** appears on *ALL* copies of the specification or portions thereof, including modifications, that you make.

THIS SPECIFICATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDER MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR

WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SPECIFICATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. COPYRIGHT HOLDER WILL BEAR NO LIABILITY FOR ANY USE OF THIS SPECIFICATION.

The name and trademarks of copyright holder may *NOT* be used in advertising or publicity pertaining to the specification without specific, written prior permission. Title to copyright in this specification and any associated documentation will at all times remain with copyright holder.

End of MNG-LC Specification.