AniPNG (Animated Portable Network Graphics) Format Version 0.7

Editor

- Glenn Randers-Pehrson, Contractor, Dynamic Science, Inc.
  glennrp at arl.mil or glennrp at gmail.com.

Status of this Memo

This is a **DRAFT proposal**. Some version of this document will become version 1.0.

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

- png-mng-misc at lists.sf.net
- png-group at w3.org

Distribution of this memo is unlimited.

At present, the latest version of this document is available on the World Wide Web from

`ftp://ftp.simplesystems.org/pub/png-group/documents/.`

Abstract

This document defines the AniPNG (Animated Portable Network Graphics) format.

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

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

The AniPNG format was derived from the MNG-1.0 format [MNG] by deletion of a number of chunks, elimination of the "simplicity profile", and by addition of the ReCO and PlAY chunks. The AniPNG format uses the same chunk structure that is defined in the PNG specification, and it shares other features of the PNG format. Any AniPNG decoder must be able to decode PNG and JNG datastreams.

Table of Contents

1 Introduction

This specification defines the format of an AniPNG (Animated Portable Network Graphics) format.

Note: This specification depends on the PNG (Portable Network Graphics) [PNG] and the JNG (JPEG Network Graphics) [JNG] specifications. It was derived from the MNG (Multiple-image Network Graphics) specification [MNG] by deletion of a number of features and by addition of the PlAY and ReCO chunks. The PNG, JNG, and MNG specifications are available at the PNG web site,

`http://www.libpng.org/pub/png/`

AniPNG is a multiple-image member of the PNG format family that can contain

- animations,
- slide shows, or
- complex still frames,

comprised of multiple PNG or JNG single-image datastreams.

Like PNG, an AniPNG datastream consists of an 8-byte signature, followed by a series of chunks. It begins with the MHDR chunk and ends with the MEND chunk. Each chunk consists of a 4-byte data length field, a 4-byte chunk type code (e.g., “MHDR”), data (unless the length is zero), and a CRC (cyclical redundancy check value).

An AniPNG datastream describes a sequence of zero or more single frames, each of which can be composed of zero or more embedded images or directives to show previously defined images.

The embedded images can be PNG or JNG datastreams.

A typical AniPNG datastream consists of:

- The 8-byte AniPNG signature.
- The MHDR chunk.
- Frame definitions. A frame is one or more layers, the last of which has a nonzero interframe delay, composited against whatever was already on the display.
- Layer definitions.
  - An embedded potentially visible image, described by PNG or JNG datastreams (a foreground layer).
  - An image that is generated from a stored object as directed by the AniPNG PlAY chunk (a foreground layer).
  - The background (a background layer).
- PlAY chunks.

- LOOP-ENDL chunks.
- SEEK chunks that mark points in the datastream where processing can be restarted.
- Various chunks for creating and manipulating objects.
- The MEND chunk.

AniPNG 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.

AniPNG is pronounced "ani-ping."

When an AniPNG 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 AniPNG datastream begins with an 8-byte signature containing

```
 138  77  78  71  13  10  26  10  (decimal)
  8a  4d  4e  47  0d  0a  1a  0a  (hexadecimal)
\212   M   N   G  \r  \n \032 \n  (ASCII C notation)
```

which is identical to the MNG-1.0 signature and 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 AniPNG employ the same mechanism as PNG, but with rules that are explained more fully (see below, Chapter 6). An AniPNG editor is not permitted to move unknown chunks across the SAVE and SEEK chunks, across any chunks that can cause images to be created or displayed, or into or out of a IHDR-IEND or similar sequence.

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 AniPNG datastream or of any embedded object is an independent entity, i.e., no chunk is ever enclosed in the data segment of another chunk.

AniPNG-compliant decoders are required to recognize and decode independent PNG or JNG datastreams.

All AniPNG datastreams are valid MNG-1.0 datastreams, although older MNG decoders might not recognize the ReCO and PlAY chunks, which were defined in 2007 and have not been registered by the PNG Registration Authority.

Because the embedded objects making up an AniPNG are normally in PNG format, AniPNG shares the good features of PNG:

- It is believed to be 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 both binary and alpha-channel transparency.
- 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, AniPNG can accommodate JPEG-encoded images that are encoded in the PNG-like JNG format).

In addition it has these features of the MNG format:

- It provides animation with variable interframe delays.
- It allows composition of frames containing multiple images.
- It facilitates the use of images as "sprites" or groups of images as "animated sprites" that can be reused in subsequent frames.
- Once an object has been decoded, any further operations with that object can be accomplished with block-transfer operations. AniPNG requires no pixel operations on decoded images.
- It capitalizes on frame-to-frame similarities to reduce the amount of data that must be included in a datastream.
- It provides "restart" points at which processing can be safely resumed in case of data loss or corruption, or to which applications can jump if they have random access to the file.
- Images and frames can be given names, allowing authors to mark them for export outside the scope of AniPNG, where they can be used for icons or similar purposes.
- A series of PNG and JNG images can be losslessly converted to AniPNG and back to a series of equivalent PNG or JNG images.
- Multiple-image GIF files, including those using the `dispose_to_previous` method, can be losslessly converted to AniPNG.

- It is complementary to MPEG and does not attempt to replace MPEG for lossy storage of video. AniPNG does, however, provide the capability of storing animations consisting of JPEG-encoded images that have been wrapped in the JNG format.

2 Terminology

See also the glossary in the PNG specification.

requirement levels
The words “MUST”, “MUST NOT”, “REQUIRED”, “SHOULD”, “SHOULD NOT”, “ReCOMMENDED”, and “OPTIONAL” in this document, which are to be interpreted as described in RFC-2119. 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.

animation
A sequence of images meant to be played at a framing rate that will give the impression of motion. We use the more generic term “sequence” to include any group of images meant to be played at some specified framing rate or under user control, not necessarily an animation, such as a slide show, as well as animations.

cacheable
A loop is “cacheable” if it produces the identical set of scenes for each iteration of the loop. Thus the scenes can be placed in a cache and replayed rather than having to be rebuilt each time.

cheap transparency
Image transparency data conveyed via the PNG tRNS chunk rather than via a full alpha channel.

clipping boundaries
Limits within which a pixel must fall to be displayed. The left and top boundaries are inclusive, while the right and bottom boundaries are exclusive.

color encoding
File gamma and chromaticity values, an sRGB rendering intent, an iCCP profile, or whatever is involved in mapping between RGB values and colors.

embedded object or image
An image that appears in-line in an AniPNG datastream.

frame
A composition of zero or more layers that have zero interframe delay time followed by a layer with a specified nonzero delay time or by the MEND chunk. A frame is to be displayed as a still picture or as part of a sequence of still images or an animation. An animation would ideally appear to a perfect observer (with an inhumanly fast visual system) as a sequence of still frames.

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.

When the frame includes the background layer, and the background layer is transparent, the transparent background is composited against the outside world and the subsequent layers are composited against the result 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 and PlAY chunks would still be measured from this offscreen origin.

framing rate

The rate, measured in frames per second, at which frames are displayed on the output device. In an AniPNG datastream, the framing rate is the interframe delay, in ticks, divided by the number of ticks per second, from the MHDR chunk. The FRAM chunk can be used to change the framing rate for a portion of the datastream.

frozen object

An object whose set of object attributes and whose object buffer are not allowed to be discarded, replaced, or modified.

image N or object N

Shorthand for "the object with the set of object attributes pointed to by 'object_id=N'".

interframe delay

The amount of time a layer should be visible when a sequence of frames or an animation is played. A layer with a zero interframe delay is combined with the subsequent layer or layers to form a frame; the frame is completed by a layer with a nonzero interframe delay or by the MEND chunk. 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 layer at the beginning of the sequence. Every frame has an interframe delay, including the frame that is displayed last; its delay is not really "interframe", nevertheless it is the amount of time that the frame should be visible before it is removed or replaced with something else.

iteration

One cycle of a loop. In this document, as is customary among computer programmers, the number of iterations of a loop includes the first cycle. A loop can have zero iterations, which means it is not executed at all.

layer

One of

- A visible embedded image, located with respect to the frame boundaries and clipped with respect to the layer clipping boundaries and the image's own clipping boundaries.

- Part of a stored image that is displayed in response to a PlAY chunk directive, located and clipped.
- 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.
- The background image, clipped, located, and displayed against a solid rectangle filled with the background color and clipped to the subframe boundaries, that is used as a background 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.

The layers in an AniPNG datastream are gathered into one or more subframes for convenience in applying frame parameters to a subset of the layers (see the definition of "subframe" below).

An embedded visible PNG or JNG datastream generates a single layer, even though it might be interlaced or progressive. If the background consists of both a background color and a background image, these are combined into a single layer.

nullify
: To nullify a chunk is to undo its effect, restoring the datastream to the condition it would have had if the chunk being nullified had never appeared.

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.

object attributes
: Properties of an object such as its existence, potential visibility, location, clipping boundaries, and a pointer to an object buffer. See Object attributes, below.

object buffer
: A 2D array of pixels, each of which has color and transparency information. See Object buffers, below.

potentially visible image
: One of

- a not-yet-defined object that is "marked", by setting its `do_not_show` flag to zero, for on-the-fly display while the embedded image that defines it is being decoded.
- an existing object that has been made potentially visible, by setting its `do_not_show` flag to zero.

prologue segment
: The first segment, when there is more than one segment.

regular segment
Any segment other than the first (also the first segment, when there is only one).

segment
A part of an AniPNG datastream starting with the MHDR chunk or with a SEEK chunk and extending to just before the next SEEK chunk (or the MEND chunk if there is no next SEEK chunk). The MHDR, MEND, SAVE, SEEK, and TERM chunks are not considered to be a part of any segment.

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 subset of the layers defined by an AniPNG datastream, gathered for convenience in applying frame parameters (i.e., clipping information, interframe delay, timeout, termination condition, and a name. See the definition of "frame" above). The extent of a subframe depends on the framing mode; it can be

- a single layer,
- the set of layers appearing between FRAM chunks,
- a background layer and a single foreground layer, or
- a background layer plus the set of layers appearing between FRAM chunks.

See the FRAM chunk specification below (Paragraph 4.3.2).

tile
A "tile" is a pixel array copied from an object and used in the PlAY chunk to construct a layer. Also, the entire background image is a tile when it is used multiple times to fill the background by the BACK chunk.

viewable image
A stored object or embedded object that is capable of being made visible. All images are viewable, once they have been decoded from a PNG or JNG datastream.

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

3 Objects

An "object", which is identified by an `object_id`, is a PNG or JNG image. The `object_id` is an unsigned sixteen-bit number that serves as the identifier of a set of object attributes.

An “image” is a viewable object.

Object 0 is a special object whose pixel data is not available for later use (see below).

3.1 Embedded objects

An embedded object is:

- A PNG datastream (IHDR, PNG chunks, IEND).
- A JNG datastream (JHDR, JNG chunks, IEND).

3.2 Object attributes

Objects have *object attributes* that can be defined and modified by the contents of various AniPNG chunks. Decoders are responsible for keeping track of them. Object attributes include:

Existence

A nonzero object comes into existence when

- a DEFI chunk creates it.

A nonzero object ceases to exist when it does not have the “frozen” attribute and

- it is the subject of a DISC chunk.
- an empty DISC chunk appears.
- a SEEK chunk appears.
- the MEND chunk appears (or the IEND chunk appears in a simple PNG or JNG file).
- a new embedded object with the same `object_id` replaces it without an intervening DEFI chunk. In this case, the new object inherits the set of object attributes from the previous object with the same `object_id`.

Object 0 always exists.

Pointer to an object buffer

Every object (except for object 0) has an object buffer. The representation of a pointer is decided by the application; pointers never appear explicitly in an AniPNG datastream. Decoders can also create an object buffer for object 0, if that is more convenient, but the information in that buffer cannot be depended upon to exist after the image has been displayed, nor can that buffer become “frozen”.

Frozen or not frozen

All objects are initially “not frozen”. Any objects in existence (except for object 0) when the SAVE chunk is encountered become “frozen”, along with the object buffers that they point to.

Potential visibility

The “potential visibility” of an object is determined by the `do_not_show` byte of the DEFI chunk that introduced it. When an embedded object is “potentially visible,” it can be displayed “on-the-fly” as it is being decoded.

Viewability.

An object is viewable if it has a viewable object buffer. It is nonviewable if it has a nonviewable object buffer or if its object buffer has not yet been created or has been destroyed. Any attempt to display a nonviewable object must be ignored and not treated as an error.

A nonviewable object becomes viewable immediately when the decoder receives a viewable object buffer, and if the object is potentially visible it can be displayed “on-the-fly” while the object buffer is being decoded or updated. Note that object 0 is only viewable while its embedded image is being decoded and displayed on-the-fly, after which it becomes nonviewable again because no object buffer is ever created for object 0.

Location

The X and Y location of an object is determined by the DEFI chunk that introduced it and can be modified by the PlAY chunk. It is permitted to change the location of “frozen” objects, provided that the encoder includes a DEFI or PlAY chunk prior to the end of the segment that restores their locations to their “saved” positions.

Clipping boundaries

The clipping boundaries of an object are determined by the DEFI chunk that introduced it and can be modified by the PlAY chunk. It is permitted to change the clipping boundaries of “frozen” objects, provided that the encoder includes a DEFI chunk or PlAY chunk prior to the end of the segment that restores the boundaries to their “saved” values.

Additional information

While not required by this specification, applications may wish to store other information about the object, for error-checking or other purposes outside the scope of this specification.

3.3 Object buffers

An object buffer is created by the appearance of an embedded object in the datastream, with a nonzero `object_id`, or by the first layer created after the ReCO chunk begins recording. The contents of an object buffer can be modified by decoding a new embedded object with the same `object_id` or by creating a layer while the ReCO chunk is recording.

Object buffers contain a 2D array of pixel data and can contain additional information. In addition, decoders are responsible for keeping track of some properties of the data in the object buffer:

Object 0 conceptually never has an object buffer. Decoding applications can create one for their own convenience, but such an object buffer must never be made available to the rest of the AniPNG datastream or be considered viewable after it has been processed.

Viewability of object buffer

Any object that points to a viewable object buffer can be displayed, but one that points to a nonviewable one cannot. Any attempt to do so must be ignored.

- A PNG or JNG datastream always has the "viewable" attribute.

Format of data in the object buffer

The data can be stored by the decoder in its original form or in any form that is convenient, such as an X Window System "pixmap", even though that form might not have sufficient 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. Similarly, a JNG image could be stored in the same form, after the pixels have been decoded, converted to RGB form, and gamma-corrected.

Frozen or not frozen

All object buffers are initially "not frozen". Any object buffers in existence when the SAVE chunk is encountered become "frozen". Decoders do not actually have to store this flag except as a sanity check, because they can depend on the fact that a "frozen" object buffer will always have at least one "frozen" object whose "buffer pointer" points to it.

3.4 Object 0

Object 0 is a special object that has a set of object attributes that control its location, clipping, and visibility properties, but does not have an object buffer. The object attributes, which can be modified by the DEFI chunk, are applied to subsequent embedded objects whose `object_id` is zero. The pixel data for object 0 is available only for on-the-fly display and not available for later use. If at the end of any segment the attribute values are different from the default/saved values, they become undefined when a SEEK chunk appears.

4 AniPNG Chunks

This chapter describes chunks that can appear at the top level of an AniPNG datastream.

Chunk structure (length, name, data, CRC) and the chunk-naming system are identical to those defined in the PNG specification [PNG]. As in PNG, all integers that require more than one byte must be in network byte order.

Unlike PNG, fields can be omitted from some AniPNG chunks with a default value if omitted. This is permitted only when explicitly stated in the specification for the particular chunk. If a field is omitted, all the subsequent fields in the chunk must also be omitted and the chunk length must be shortened accordingly.

4.1 Critical AniPNG control chunks

This section describes critical AniPNG control chunks that AniPNG-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 AniPNG editors.

4.1.1 MHDR AniPNG datastream header

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

The MHDR chunk contains 28 bytes, none of which can be omitted:

```
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).
Unused:               4 bytes:(ignored).
```

Decoders can ignore the “informative” `nominal_frame_count`, `nominal_layer_count`, and `nominal_play_time` fields.

The `frame_width` and `frame_height` fields give the intended display size (measured in pixels) and provide default clipping boundaries (see Recommendations for encoders, below). It is strongly recommended that these be set to zero if the AniPNG datastream contains no visible images.

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 a sequence of images. 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 the sequence of frames 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 `nominal_layer_count` field contains a zero, the layer count is unspecified. If it is nonzero, it contains the number of layers, including all background layers, when the file is displayed 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-byte 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.

The final four bytes of the MHDR chunk are unused. They are for compatibility with MNG-1.0 decoders, and must be ignored by AniPNG decoders. AniPNG encoders should write the 32-bit unsigned integer "2015" in this field to indicate that any AniPNG features might be present, or "991" if the PlAY and ReCO chunks are not present. If they wish, they may write any other appropriate MNG-1.0 "simplicity profile" as defined in the MNG-1.0 specification [MNG], but this is not required.

4.1.2 MEND End of AniPNG datastream

The MEND chunk's data length is zero. It signifies the end of an AniPNG datastream that began with the MHDR chunk. AniPNG datastreams that begin with the IHDR or JHDR chunk are terminated by the IEND chunk instead.

4.1.3 LOOP, ENDL Define a loop

The LOOP chunk provides a "shorthand" notation that can be used to avoid having to repeat identical chunks in an AniPNG datastream.

Its contents are the first two or more of the following fields. If any field is omitted, all subsequent fields must also be omitted:

```
Nest_level:       1 byte (unsigned integer).
Iteration_count: 4 bytes (unsigned integer),
                    range [0..2^31-1].
Termination_condition:
                  1 byte (unsigned integer).
                    Must be omitted if
                    termination_condition=0, which
                    means Deterministic, not cacheable,
                    or if iteration_count=0.
                    1: Decoder discretion, not cacheable.
```

```
                    2: User discretion, not cacheable.
                    3: External signal, not cacheable.
                    4: Deterministic, cacheable.
                    5: Decoder discretion, cacheable.
                    6: User discretion, cacheable.
                    7: External signal, cacheable.
Iteration_min:   4 bytes(unsigned integer).  Must be
                    present if termination_condition is
                    3 or 7.  If omitted, the default
                    value is 1.
Iteration_max:   4 bytes (unsigned integer).  Must be
                    present if termination_condition is 3
                    or 7; must be omitted if iteration_min
                    is omitted; if omitted, the default
                    value is infinity.
Signal_number:   4 bytes (unsigned integer).  Must be
                    present if termination_condition is 3
                    or 7.  Must not be present otherwise.
Additional
  signal_number: 4 bytes.  May be present only if
                    termination_condition is 3 or 7.
...etc...
```

Decoders must treat the chunks enclosed in a loop exactly as if they had been repeatedly spelled out. The `termination_condition` field can be used to inform decoders that it is safe to change the number of loop iterations.

Simple decoders can ignore all fields except for the `iteration_count`.

When the LOOP chunk is present, an ENDL chunk with the same `nest_level` must be present later in the AniPNG datastream. Loops can be nested. Each inner loop must have a higher value of `nest_level` than the loop that encloses it, though not necessarily exactly one greater.

The termination condition specifies how the actual number of iterations is determined. It is very similar to the termination condition field of the FRAM chunk, and can take the same values:

Deterministic

This is the default behavior, when the `termination_condition` field is omitted or has a value that is unrecognized by the decoder. The loop terminates after exactly the number of iterations specified by the iteration count. This value must be used if altering the number of repetitions would mess up the AniPNG datastream, but can be used merely to preserve the author's intent.

Decoder-discretion

The number of iterations can be chosen by the decoder, and must not be less than `iteration_min` nor more than `iteration_max`. If the decoder has no reason to choose its own value, it should use the `iteration_count`. One example of a decoder wishing to choose its own value is a real-time streaming decoder hovering at a loop while waiting for its input buffer to fill to a comfortable level.

User-discretion

The number of iterations should be chosen by the user (e.g., by pressing the <escape> key), but the decoder must enforce the `iteration_min` and `iteration_max` limits. Some decoders might not be able to interact with the user, and many decoders will find that nested user-discretion loops present too great of a user-interface challenge, so the <user-discretion> condition will probably usually degenerate into the <decoder-discretion> condition.

External-signal

The number of iterations must not be less than `iteration_min` nor more than `iteration_max`. The exact number can be determined by the arrival of a signal whose number matches one of the `signal_number` fields.

When the value of the `termination_condition` field is 4 or more, the loop is guaranteed to be "cacheable", which means that each iteration of the loop produces the same sequence of frames, and that all objects and object buffers are left in the same condition at the end of each iteration. Decoders can use this information to select a different strategy for handling the loop, such as storing the composited frames in a cache and replaying them rather than decoding them repeatedly.

The `iteration_min` and `iteration_max` can be omitted. If the condition is <deterministic>, the values are not used. Otherwise, defaults of 1 and <infinity> are used. The `iteration_count`, `iteration_min`, and `iteration_max` can be any non-negative integers or <infinity>, but they must satisfy `iteration_min <= iteration_count <= iteration_max`. Infinity is represented by 0x7fffffff.

If `iteration_count` is zero, the `termination_condition`, the subsequent fields must be omitted, and the loop is done zero times. Upon encountering a LOOP chunk whose `iteration_count` is zero, decoders simply skip chunks until the matching ENDL chunk is found, and resume processing with the chunk immediately following it.

The `signal_number` can be omitted only if the termination condition is not <external-signal>. There can be any number of `signal_number` fields. Signal_number=0 is reserved to represent any input from a keyboard or pointing device, and 1–255 are reserved to represent the corresponding character code, received from a keyboard or simulated keyboard, and values 256–1023 are reserved for future definition by this specification.

An infinite or just overly long loop could give the appearance of having locked up the machine. Therefore a decoder should always provide a simple method for users to escape out of a loop or delay, either by abandoning the AniPNG entirely or just proceeding to the next SEEK chunk (the SEEK chunk makes it safe for a viewer to resume processing after it has jumped out of the interior of a segment).

AniPNG editors that extract a series of PNG or JNG files from an AniPNG datastream are expected to execute the loop only `iteration_min` times, when the termination condition is not <deterministic>.

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

```
Nest_level: 1 byte (unsigned integer), range [0..255].
```

When the ENDL chunk is encountered, the loop iteration count is decremented, if it is not already zero. 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 `nest_level` must be present earlier in the AniPNG datastream. See below. Loops must be properly nested: if a LOOP chunk with higher `nest_level` appears inside a LOOP/ENDL pair, a matching ENDL chunk must also appear to close it.

The SAVE and SEEK chunks are not permitted inside a LOOP-ENDL pair. To rerun an entire datastream that includes these chunks, use the TERM chunk instead. See below (Paragraph 4.2.7).

4.2 Critical AniPNG image defining chunks

The chunks described in this section create objects and may cause them to be immediately displayed.

4.2.1 DEFI Define an object

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

The DEFI chunk contains 2, 3, 4, 12, or 28 bytes. If any field is omitted, all subsequent fields must also be omitted.

```
Object_id:     2 bytes (unsigned integer) identifier to be
                 given to the objects that follow the DEFI
                 chunk.

Do_not_show:  1 byte (unsigned integer)
                 0:  Make the objects potentially visible.
                 1:  Make the objects not potentially
                     visible.

Concrete_flag: 1 byte (unsigned integer)
                 0:  Make the objects "abstract" .
                 AniPNG decoders can ignore this flag.
                 AniPNG encoders should write "0".

X_location:    4 bytes (signed integer).
                 The X_location and Y_location fields can
                 be omitted as a pair.

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).
```

If the object number for an object is nonzero, subsequent chunks can use this number to identify it.

When the object number for an object is zero, its object buffer can be discarded immediately after it has been processed. Its object attributes must be retained.

Negative values are permitted for the X and Y location and clipping boundaries. The left and top boundaries are inclusive, while the right and bottom boundaries are exclusive. The positive directions are downward and rightward from the frame origin (see Recommendations for encoders, below).

Multiple IHDR-IEND and JHDR-IEND objects can follow a single DEFI chunk. When `object_id` is nonzero, the DEFI chunk values remain in effect until another DEFI chunk or a SEEK chunk appears. The `object_id` can only be changed by using another DEFI chunk.

If no DEFI chunk is in effect (either because there is none in the datastream, or because a DISC or SEEK chunk has caused it to be discarded), 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
```

The object attributes for all existing unfrozen objects except for object 0 become undefined when a SEEK chunk is encountered.

The object attributes for object 0 become undefined when a SEEK chunk is encountered, only if they have been reset to values other than these defaults. It is the encoder's responsibility to reset them explicitly to these values prior to the end of every segment in which they have been changed, or to include a full DEFI chunk prior to embedding object 0 in any segment.

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.

A set of object attributes is created or an existing one is modified when the DEFI chunk appears, but an object buffer is neither created nor discarded. If `object_id` is an identifier that already exists when a

DEFI chunk appears, the set of object attributes (except for the pointer to the object buffer) is immediately replaced. The contents of the object buffer do not change, however, until and unless an IHDR or JHDR chunk is encountered. When one of these chunks appears, all of the contents of the object buffer previously associated with the identifier are discarded and the new data is stored in the object buffer.

4.2.2 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.

4.2.3 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.

The IHDR and IEND chunks and any chunks between them are written and decoded according to the PNG specification, except as extended in this section. These extensions do not apply to standalone PNG datastreams that have the PNG signature, but only to PNG datastreams that begin with the AniPNG signature or are embedded in an AniPNG datastream that begins with an AniPNG signature.

- An additional PNG filter method is defined:

  ```
  64: Adaptive filtering with five basic types and
      intrapixel differencing.
  ```

  The intrapixel differencing transformation, which is a modification of a method previously used in the LOCO image format [LOCO], is

  ```
  S0 = Red  - Green (when color_type is 2 or 6)
  S1 = Green        (when color_type is 2 or 6)
  S2 = Blue - Green (when color_type is 2 or 6)
  S3 = Alpha        (when color_type is 6)
  ```

in which S0-S3 are the samples to be passed to the next stage of the filtering procedure.

The transformation is done in integer arithmetic in sufficient precision to hold intermediate results, and the result is calculated modulo $2^{\text{sample\_depth}}$. Intrapixel differencing (subtracting the green sample) is only done for color types 2 and 6, and only when the filter method is 64. This filter method is not permitted in images with color types other than 2 or 6.

Conceptually, the basic filtering is done after the intrapixel differencing transformation has been done for all pixels involved in the basic filter, although in practice the operations can be combined.

To recover the samples, the transformation is undone after undoing the basic filtering, by the inverse of the intrapixel differencing transformation, which inverse is

```
Red   = S0 + S1
Green = S1
Blue  = S2 + S1
Alpha = S3
```

As in the forward transformation, the inverse transformation is done in integer arithmetic in sufficient precision to hold intermediate results and the result calculated modulo $2^{\text{sample\_depth}}$.

Applications that convert an AniPNG datastream to a series of PNG datastreams must convert any PNG datastream with the additional filter method 64 to a standard PNG datastream with a PNG filter method (currently 0 is the only valid filter method).

Applications must not write independent PNG datastreams with the PNG signature and .png file extension with the new filter method, until and unless it should become officially approved for use in PNG datastreams. They may write independent PNG datastreams with the AniPNG signature and .mng file extension with the new filter method.

- If a global PLTE chunk appears in the top-level AniPNG 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. AniPNG 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.

- If the PNG sRGB, gAMA, iCCP, or cHRM chunks appear in the top-level AniPNG datastream (and have not been nullified), but none of them appear in the PNG datastream, then the values are inherited from the top level as though the chunks had actually appeared in the PNG datastream. Data from such chunks appearing in the PNG datastream take precedence over the inherited values. If any one of these chunks, or any chunk in a future version of this specification that defines the color space, appears in the PNG datastream, none of them is inherited. AniPNG applications that recreate PNG files must write these chunks, if they are inherited, in the output PNG files. If the sRGB chunk is present in an AniPNG datastream, it need not be accompanied in the AniPNG datastream by gAMA and cHRM chunks, despite the recommendation in the PNG specification. Any AniPNG viewer that processes the gAMA chunk must also recognize and process the sRGB chunk. It can treat it as if it were a gAMA chunk containing the value .45455 and it can ignore its “intent” field.

Note that the top-level color space chunks are used only to supply missing color space information to subsequent embedded PNG or JNG datastreams. They do not have any effect on already-decoded objects.

- If the PNG sPLT chunk appears in the top-level AniPNG datastream, it takes precedence over any sPLT chunk appearing in the PNG datastream. AniPNG applications that recreate PNG files should not copy top-level sPLT chunks to the output PNG files, because a suggested palette for rendering a group of images is not necessarily the best palette for rendering a single image.
- The PNG oFFs and pHYs chunks and any chunks in a future version of this specification that attempt to set the pixel dimensions or the drawing location must be ignored by AniPNG viewers and simply copied (according to the copying rules) by AniPNG editors.
- The PNG gIFg, gIFt, and gIFx chunks must be ignored by viewers and must be copied according to the copying rules by AniPNG editors.

If `object_id` is zero, there is no need to store the pixel data after decoding it and perhaps displaying it.

If an object already exists with the same `object_id`, the contents of its object buffer are replaced with the new data.

4.2.4 JHDR, JNG chunks, IEND

A JNG (JPEG Network Graphics) datastream.

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

The JHDR and IEND chunks and any chunks between them are written and decoded according to the JNG specification.

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

4.2.5 ReCO Record current frame

The ReCO chunk is useful for accomplishing the equivalent of the GIF “restore-to-previous” disposal method.

This chunk is currently not a registered MNG chunk. Registration is being pursued with the PNG Registration Authority. In the meantime, the private form of the chunk name (ReCO, with a lowercase “e”), must be used in any public datastreams.

It has two fields

```
   object_id  (16-bit unsigned int):  existing object to be used
for
                     storing image of frame
   mode (byte)     0: initialize object to transparent and start
                     recording offscreen copy of current frame
                     (including any background layers).
                  1: stop recording.
                  2: restart recording with existing contents.
```

The `object_id` must already have been the subject of a DEFI chunk and must still exist. The dimensions and location of the area to be recorded are specified in the DEFI chunk.

Each layer that is defined while the ReCO chunk is in recording mode is composited over (or replaces, depending on the `framing_mode`) whatever is in the object.

It is permitted to have more than one ReCO object open at the same time. The first ReCO chunk for each object in a segment must have mode 0, and the last ReCO chunk for each object in a segment must have mode 1.

4.2.6 DISC Discard objects

The DISC chunk can be used to inform the decoder that it can discard the object data associated with the associated object 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 object identifiers. The number of objects to be discarded is the chunk's data length, divided by two.

```
Discard_id: 2 bytes (nonzero unsigned integer).
...etc...
```

If the DISC chunk is empty, all nonzero objects except those preceding the SAVE chunk (i.e., except for the "frozen" objects) can be discarded. If a SAVE chunk has not been encountered, all objects can be discarded. Note that each appearance of a SEEK chunk in the datastream implies an empty DISC chunk.

If the DISC chunk is not empty, the listed objects can be discarded.

When an object is discarded, any location, potential visibility, and clipping boundary data associated with it is also discarded.

It is not an error to include an `object_id` in the `discard_id` list, when no such object has been stored, or when the object has already been discarded.

It is an error to name explicitly any "frozen" object in the DISC list.

4.2.7 TERM Termination action

The TERM chunk suggests how the end of the AniPNG 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 indefinitely.
                           3: Repeat the sequence starting
                              immediately after the TERM
                              chunk and ending with the
                              MEND 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 indefinitely.

                          This and the subsequent 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
                          sequence.

Iteration_max:           4 bytes (unsigned integer) Maximum
                         number of times to execute the
                         sequence.  Infinity is represented
                         by 0x7fffffff.
```

The final frame of the sequence endures at least for the maximum of:

- The value from the “delay” field of the TERM chunk, if the “delay” field is present
- The interframe delay of the final frame, as specified in the FRAM chunk
- One tick, as specified in the MHDR chunk

before the sequence is repeated or before ceasing displaying anything.

If no TERM chunk is present, viewers should assume `termination_action=0` as the default.

The loop created by processing a TERM chunk must always be treated by the decoder as if it were a cacheable <user-discretion> loop, with `iteration_min=1`.

Applications must not depend on anything that has been drawn on the output buffer or device during the previous iteration. Its contents become undefined when the TERM loop restarts.

AniPNG editors that extract a series of PNG or JNG files from an AniPNG datastream are expected to execute the TERM loop only once, rather than emitting the files repeatedly.

The TERM chunk, if present, must appear either immediately after the MHDR chunk or immediately prior to a SEEK chunk. The TERM chunk is not considered to be a part of any segment for the purpose of determining the copy-safe status of any chunk. Only one TERM chunk is permitted in an AniPNG datastream.

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

4.3 Critical AniPNG image displaying chunks

The chunks in this section cause existing objects and embedded objects to be displayed on the output device, and control their location, clipping, and timing and the background against which they are displayed.

4.3.1 BACK Background

The BACK chunk suggests or mandates a background color, image, or both against which transparent, clipped, or less-than-full-frame images can be displayed. This information will be used whenever the application subsequently needs to insert a background layer, unless another BACK chunk provides new background information before that happens.

The BACK chunk contains 6, 7, 9, or 10 bytes. If any field is omitted, all subsequent fields must also be omitted.

```
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 and background
                         image are advisory.  Applications
                         can use them if they choose to.
                      1: Background color is mandatory.
                         Applications must use it.
                         Background image is advisory.
                      2: Background image is mandatory.
```

```
                         Applications must use it.
                         Background color is advisory.
                      3: Background color and background
                         image are both mandatory.
                         Applications must use them.
                      This byte can be omitted if the
                      subsequent fields are also omitted.
                      If so, the background color is
                      advisory.

Background_image_id:
                   2 bytes (unsigned nonzero integer).
                   Object_id of an image that is to be used
                   as the background layer or part of it.
                   If the image does not cover the area
                   defined by the layer clipping boundaries
                   with opaque pixels, the remainder of
                   this area is filled with the background
                   color or application background and the
                   background image is composited against
                   it.  This field can be omitted if the
                   background_tiling byte is also omitted;
                   if so, no background image is defined,
                   and the background image_id from any
                   previous BACK chunk becomes undefined.

Background_tiling:
                   1 byte (unsigned integer).
                      0: Do not tile the background.
                      1: Tile the background with the
                          background image.

                   This field can be omitted; if so, do not
                   tile the background.
```

The first layer displayed by a viewer is always a background layer that fills the entire frame. The BACK chunk provides a background that the viewer can use for this purpose (or must use, if it is mandatory). If it is not "mandatory" the viewer can choose another background if it wishes. If the BACK chunk is not present, or if the background is not fully opaque or has been clipped to less than full frame, the viewer must provide or complete its own background layer for the first frame. Each layer after the first must be composited over the layers that precede it, until a FRAM chunk with framing mode 3 or 4 causes another background layer to be generated.

Viewers are expected, however, to composite every foreground layer against a fresh copy of the background, when the framing mode given in the FRAM chunk is 3, and to composite the first foreground layer of each subframe against a fresh copy of the background, when the framing mode is 4. Also, when the framing mode is 3 or 4 and no foreground layer appears between consecutive FRAM chunks, a background layer alone is displayed as a separate frame.

The images and the background are both clipped to the subframe boundaries given in the FRAM chunk. Anything outside these boundaries is inherited from the previous subframe. If the background layer is transparent and the subsequent foreground layers do not cover the transparent area with opaque pixels, the application's background becomes re-exposed in any uncovered pixels within the subframe boundaries.

The background image (or tiled assembly) is also clipped to its own boundaries and located like any other image, and is only displayed if it is potentially visible. When the background image is used for tiling, the upper left tile is located according to the background image's location attributes and the entire assembly is clipped according to its clipping attributes. Viewers might actually follow some other procedure, but the final appearance of each frame must be the same as if they had filled the area within the subframe boundaries with the background color, then displayed the background image, and then displayed the foreground image (or images), without delay.

Note that any background layer, including the one that begins the first frame of the datastream, must be inserted at the latest possible moment, in case a new BACK chunk appears, before that moment.

It is not an error to specify a `background_image_id` when such an image is not viewable and potentially visible or does not yet exist or ceases to exist for some reason, or to fail to specify one even when the `mandatory_background` flag is 2 or 3. Viewers must be prepared to fall back temporarily to using the background color or application background in this event, and to resume using the background image whenever a potentially visible viewable object with the `background_image_id` becomes available. They also must be prepared for the contents, viewability, location, potential visibility, and clipping boundaries of the background image to change, just like any other object, if it has not been "frozen". The background image is allowed to have transparency.

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 the RGB color samples {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 AniPNG datastream. If no such chunks appear, the color space is unknown.

The color space of the background image, if one is used, is determined in the same manner as the color space of any other image.

The data from the BACK chunk takes effect the next time the decoder needs to insert a background layer, and remains in effect until another BACK chunk appears.

For the purpose of counting layers, when the background consists of both a background color and a background image, these are considered to generate a single layer and there is no delay between displaying the background color and the background image.

Multiple instances of the BACK chunk are permitted in an AniPNG datastream.

The BACK chunk can be omitted. If a background is needed and the BACK chunk is omitted, the background layer is a transparent rectangle of dimensions `frame_width` by `frame_height` and the viewer must supply its own background. For the purpose of counting layers, such an implicit background layer is counted the same as a background supplied by the BACK chunk.

In practice, most applications that use AniPNG 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.

4.3.2 FRAM Frame definitions

The FRAM chunk provides information that a decoder needs for generating frames and interframe delays. 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.

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 ("upcoming subframe" refers to the subframe immediately following the FRAM chunk). 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.

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_mode:  1 byte.

                    0:  Do not change framing mode.

                    1:  No background layer is generated,
                        except for one ahead of the very
                        first foreground layer in the
                        datastream.  The interframe delay
                        is associated with each foreground
                        layer in the subframe.

                    2:  No background layer is generated,
                        except for one ahead of the very
                        first image in the datastream.  The
                        interframe delay is associated only
                        with the final layer in the subframe.
                        A zero interframe delay is associated
                        with the other layers in the
                        subframe.

                    3:  A background layer is generated ahead
                        of each foreground layer.  The
                        interframe delay is associated with
                        each foreground layer, and a zero
                        delay is associated with each
                        background layer.
```

```
                  4:  The background layer is generated
                      only ahead of the first foreground
                      layer in the subframe.  The
                      interframe delay is associated only
                      with the final foreground layer in
                      the subframe.  A zero interframe
                      delay is associated with the
                      background layers, except when
                      there is no foreground layer in the
                      subframe, in which case the
                      interframe delay is associated with
                      the sole background layer.

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 the
                subsequent fields are also omitted.

Change_interframe_delay:
                1 byte.
                  0:  No.
                  1:  Yes, for the upcoming subframe only.
                  2:  Yes, also reset default.

                This field and all subsequent fields can be
                omitted as a group if no frame parameters
                other than the framing mode or the subframe
                name are changed.

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

               This field can be omitted only if the
```

```
              previous field is also omitted.

Change_layer_clipping_boundaries:
              1 byte.
                 0:  No.
                 1:  Yes, for the upcoming subframe only.
                 2:  Yes, also reset default.

              This field can be omitted only if the
              previous field is also omitted.

Change_sync_id_list:
              1 byte.
                 0:  No.
                 1:  Yes, for the upcoming subframe only.
                 2:  Yes, also reset default list.

              This field can be omitted only if the
              previous field is also omitted.

Interframe_delay:
               4 bytes (unsigned integer).  This field
                 must be omitted if the
                 change_interframe_delay field is zero
                 or is omitted.  The range is
                 [0..2^31-1] ticks.

Timeout:       4 bytes (unsigned integer).  This field
                 must be omitted if the
                 change_timeout_and_termination field is
                 zero or is omitted.  The range is
                 [0..2^31-1].  The value 2^31-1
                 (0x7fffffff) ticks represents an
                 infinite timeout period.

Layer_clipping_boundary_delta_type:
               1 byte (unsigned integer).
                 0: Layer clipping boundary values are
                    given directly.
                 1: Layer clipping boundaries are
                    determined by adding the FRAM data
                    to the values from the previous
                    subframe.
               This and the following four fields must
               be omitted if the
               change_layer_clipping_boundaries field
               is zero or is omitted.

Left_layer_cb or Delta_left_layer_cb:
```

```
                 4 bytes (signed integer).

Right_layer cb or Delta_right_layer_cb:
                 4 bytes (signed integer).

Top_layer_cb or Delta_top_layer_cb:
                 4 bytes (signed integer).

Bottom_layer_cb or Delta_bottom_layer_cb:
                 4 bytes (signed integer).

Sync_id:        4 bytes (unsigned integer).  Must be
                   omitted if change_sync_id_list=0 and
                   can be omitted if the new list is empty;
                   repeat until all sync_ids have been
                   listed.  The range is [0..2^31-1].
```

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.

Any of these events generates a layer, even if no pixels are actually changed:

- Decoding a IHDR-IEND sequence at the AniPNG level, when it defines a potentially visible image.
- Decoding a JHDR-IEND sequence at the AniPNG level, when it defines a potentially visible image.
- Decoding a layer structure of the PlAY chunk.
- Also, decoding a FRAM chunk, when the current framing mode requires a background layer (framing mode is 3 or 4) and none of the above have already caused the background layer to be inserted since the previous FRAM chunk. Such background layers must be included in the `nominal_layer_count` field of the MHDR chunk.

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, decoders 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.

Also, viewers that jump to a segment must insert a background layer, with a zero delay, ahead of the segment, even when the BACK chunk is not present in the prologue segment, if they jumped from the interior of a segment. Such layers are *not* included in either the layer count or the frame count.

Framing mode 1

When `framing_mode` is 1, the decoder must wait until the interframe delay for the previous frame has elapsed before displaying each image. Each foreground layer is a separate subframe and frame, composited over the preceding frame.

Framing mode 2

Framing mode 2 is the same as framing mode 1, except that the interframe delay occurs between subframes delimited by FRAM chunks rather than between individual layers. All of the foreground layers between consecutive FRAM chunks make up a single subframe, composited over the preceding frame.

In the usual case, the interframe delay is nonzero, and multiple layers are present, so each frame is a single subframe composed of several layers. When the interframe delay is zero, the subframe is combined with subsequent subframes until one with a nonzero interframe delay is encountered, to make up a single frame.

The decoder must wait until the interframe delay for the previous frame has elapsed before displaying the 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.

Framing mode 3

When framing_mode=3, a background layer is generated and displayed immediately before each image layer is displayed. Otherwise, framing mode 3 is identical to framing mode 1. Each foreground layer together with its background layer make up a single subframe and frame.

When the background layer is transparent or does not fill the clipping boundaries of the image layer, the application is responsible for supplying a background color or image against which the image layer is composited, and if the AniPNG is being displayed against a changing scene, the application should refresh the entire AniPNG frame against a new copy of the background layer whenever the application's background scene changes.

In effect, the composition of background and image replaces the pixels of the preceding frame that fall within the background layer's clipping boundaries, while any pixels outside those boundaries are composited over the preceding frame.

Framing mode 4

When `framing_mode=4`, the background layer is generated and displayed immediately before each frame, i.e., after each FRAM chunk, with no interframe delay before each image. The decoder must wait until the interframe delay for the previous frame has elapsed before displaying the background layer. Otherwise, framing mode 4 is identical to framing mode 2. All of the foreground layers between consecutive FRAM chunks, together with one background layer, make up a single subframe.

A transparent or clipped background layer is handled as in framing mode 3.

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. No specific use for the subframe name is specified in this document, except that it can be included in the optional index that can appear in the SAVE chunk. Applications can use this field for such purposes as constructing an external list of subframe 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 (see Uniform Resource Identifier below).

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, layers are frames. When it is zero, a frame consists of any number of consecutive subframes, until a nonzero delay subframe is encountered and completed or the MEND chunk is encountered. Decoders are not obligated or encouraged 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, regardless of the framing mode. The interframe delay field also conveys the final delay, which is the desired minimum time to elapse before replacing the final frame with something else or before starting another iteration according to the TERM chunk. The TERM chunk can increase the final delay.

The timeout field can be a number or <infinity>. Infinity can be represented by 0x7fffffff. Under certain termination conditions, the application can adjust the interframe delay, provided that it is not greater than the greater of the specified interframe delay and the timeout nor smaller than the smaller of the specified interframe delay and the timeout.

The termination condition given in the `change_timeout_and_termination` field specifies whether and over what range the normal interframe delay can be lengthened or 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 shorten or lengthen the duration of the frame, to any duration between the interframe delay and the timeout. A streaming decoder could take the opportunity to wait for its input buffer to fill to a comfortable level.

user-discretion

If the interframe delay is nonzero, the decoder should wait for permission from the user (e.g., via a keypress) before proceeding, but must wait no less than the smaller of the timeout and the interframe delay nor no longer than the greater of the timeout and the interframe delay. If the decoder cannot interact with the user, this condition degenerates into "decoder-discretion".

external-signal

If the interframe delay is nonzero, the decoder should wait for the arrival of a signal whose number matches a `sync_id`, but must wait no less than the smaller of the timeout and the interframe delay nor no longer than the greater of the timeout and the interframe delay. If the decoder cannot receive signals, this condition degenerates into "decoder-discretion".

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 layer clipping boundaries from the FRAM chunk are only used for clipping, not for placement. The DEFI and PlAY chunks can be used to specify the placement and clipping boundaries of each image within the layer. 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. The left and top subframe boundaries are inclusive, while the right and bottom boundaries are exclusive.

If the layers do not cover the entire area defined by the layer clipping boundaries with opaque pixels, 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 layer clipping boundaries for the subframe. Any pixels outside the layer clipping boundaries remain unchanged from the previous layer.

The `interframe_delay` field gives the duration of display, which is the minimum time that must elapse from the beginning of displaying one layer until the beginning of displaying the next (unless the termination condition and timeout permit this time to be shortened). 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 the layer is to be combined with the subsequent layer or layers 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 first frame 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.

Note that the rules for omitting the interframe delay, timeout, clipping boundary, and sync id fields of the FRAM chunk are different from the general rule stated in AniPNG Chunks, above (Chapter 4). These fields are either present in the chunk data or omitted from it according to the contents of the corresponding "change" byte.

4.3.3 Embedded images

In addition to creating MNG objects (see above (Paragraph 4.2.3), and above, Paragraph 4.2.4), the IHDR-IEND and JHDR-IEND chunks display them when the DEFI chunk that introduced them set `do_not_show` to zero, to specify on-the-fly display.

A viewer can choose to display the image while it is being decoded, perhaps taking advantage of the PNG interlacing method or the JNG progressive display feature, or to display it after decoding is complete.

4.3.4 PlAY Playlist

The PlAY chunk contains a compressed "play list" that provides instructions to the decoder for constructing animation layers from pieces of previously defined objects.

This chunk is currently not a registered MNG chunk. Registration is being pursued with the PNG Registration Authority. In the meantime, the private form of the chunk name (PlAY, with a lowercase "l"), must be used in any public datastreams.

The PlAY chunk contains a 1-byte header followed by one or more 32-byte layer structures. The header gives the compression method:

```
byte
 0   compression_method (byte)     0: deflate
 1   layer_structure_array[n]      (n > 0; n * 32
     (compressed)                    bytes): layer
                                     structures
```

Each layer is a tile combined with a transparent background to form a layer of size (frame_width, frame_height), from the MHDR chunk. Each layer is initialized to a transparent (RGBA(0,0,0,0)) rectangle, regardless of the contents of the BACK chunk.

The layer structure array is always compressed according to the specified compression method.

A layer is described by a structure of 32 bytes:

```
byte
 0    object     (unsigned int)   Object ID of source
                                  image
 2    delta mode (unsigned byte)  0: location and
                                     boundaries are
                                     absolute
                                  1: location and
                                     boundaries are
                                     deltas from the
                                     DEFI values for
                                     the object.
 3    update mode (unsigned byte) 0: Do not update
                                     DEFI values of
                                     location and
                                     boundaries
                                  1: Update DEFI
                                     values
 4    left       (signed integer)    Left of tile
                                     destination
 8    top        (signed integer)    Top of tile
                                     destination
12    right      (signed integer)    Right of tile
                                     destination
16    bottom     (signed integer)    Bottom of tile
                                     destination
20    x_location (signed integer)   X location
                                     of object
                                     destination
24    y_location (signed integer)   Y location
```

```
                                                  of object
                                                  destination
        28    delay      (unsigned integer)  Interframe delay
```

If the `object_id` is non-zero, the tile is taken from the already existing object buffer as it exists when the PlAY chunk is encountered. If it is zero, the tile is taken from the next instance of object 0 following the PlAY chunk. If any layer of the playlist refers to object 0, all if the layers in the playlist must refer to object 0, and all of them are taken from the single immediately following instance of object 0.

The stored values of the boundaries and location are taken from the DEFI chunk. These are used as the basis for computing current values to be used when delta_mode is 1, and the results, whether from a delta calculation or from supplied absolute values, are used to update the stored values when update_mode is 1.

The layer data identifies a rectangle (`left`,`right`,`top`,`bottom`) in the object containing pixels of a tile. The boundaries are measured from the upper left corner of the frame, after the object has been relocated to the position (`x_location`,`y_location`).

Any part of the tile outside the object must be treated as transparent. Any part of the tile which extends outside the layer must be ignored. The tile width or height may be zero, in which case the layer contains all transparent pixels.

The location and clipping boundaries from the DEFI chunk for the object are used to initialize and store these values. These are used as the basis for computing current values to be used when `delta_mode` is 1, and the results, whether from a delta calculation or from supplied absolute values, are used to update the stored values in the object attributes when `update_mode` is 1.

The delay is measured in `ticks_per_second` from the MHDR chunk, and it overrides but does not replace the value from the FRAM chunk. Even if a layer's tile is entirely outside the layer, the delay for the resulting empty layer must be handled in the same way as if it were inside the frame.

The decoder must handle the layers as specified in the FRAM chunk. This might or might not involve inserting background layers between them or coalescing zero-delay layers together into one frame, and the usual clipping to layer and frame boundaries must be done.

A playlist containing references to both object 0 and a non-zero object, and two or more PlAY chunks referring to the same instance of object 0 are error conditions, and decoders must reject such PlAY chunks.

4.4 SAVE and SEEK chunks

The SAVE chunk marks a point in the datastream at which objects are "frozen" and other chunk information is "saved". The SEEK chunk marks positions in the AniPNG datastream where a restart is possible, and where the decoder must restore the "saved" information, if they have jumped or skipped to a SEEK point from the interior of a segment. They only need to restore information that they will use, e.g., a viewer that processes gAMA and global PLTE and tRNS, but ignores iCCP and sPLT, need only restore the value of gamma and the global PLTE and tRNS data from the prologue segment but not the values of the iCCP and sPLT data.

Simple decoders that only read AniPNG datastreams sequentially can safely ignore the SAVE and SEEK chunks, although it is recommended that, for efficient use of memory, they at least mark existing objects as “frozen” when the SAVE chunk is processed and discard all “unfrozen” objects whenever the SEEK or empty DISC chunk is processed.

4.4.1 SAVE Save information

The SAVE chunk marks a point in the datastream at which objects are “frozen” and other chunk information is “saved”; a decoder skipping or jumping to a SEEK chunk from the interior of a segment must restore the “saved” chunk information if it has been redefined or discarded. In addition, the SAVE chunk can contain an optional index to the AniPNG datastream.

The SAVE chunk can be empty, or it can contain an index consisting of the following:

```
Offset_size: 1 byte (unsigned integer).
                4: Offsets and nominal start times are
                   expressed as 32-bit integers.
                8: Offsets and nominal start times are
                   expressed as 64-bit integers.
```

plus zero or more of the following index entries:

```
Entry_type:  1 byte (unsigned integer).
               0: Segment with nominal start time, nominal
                  layer number, and nominal frame number.
               1: Segment.
               2: Subframe.
               3: Exported image.
Offset:     4 or 8 bytes (unsigned integer).  Must be
              omitted if entry_type > 1, set equal to zero
              if the offset is unknown.
Nominal_start_time:
            4 or 8 bytes: (unsigned integer).  Start time
              of the segment, measured in ticks from the
              beginning of the sequence, assuming that all
              prior segments were played as intended on an
              ideal player.  Must be omitted if
              entry_type > 0.
Nominal_layer_number:
           4 bytes (unsigned integer).  Sequence number of
              the first layer in the segment, assuming
              that all prior segments were played as
              intended on an ideal player; the first layer
              of the first segment being layer 0.  Must be
              omitted if entry_type > 0.
Nominal_frame_number:
          4 bytes (unsigned integer).  Sequence number of
```

```
                 the first frame in the segment, assuming
                 that all prior segments were played as
                 intended on an ideal player; the first
                 frame of the first segment being frame 0.
                 Must be omitted if entry_type > 0.
Name:          1-79 bytes (Latin-1 text).  Must be omitted
                 for unnamed segments.  The contents of this
                 field must be the same as the name field in
                 the corresponding SEEK, FRAM, or eXPI chunk.
Separator:     1 byte (null) (must be omitted after the final
                 entry).
```

The SAVE chunk must be present when the SEEK chunk is present. It appears after the set of chunks that define information that must be retained for the remainder of the datastream. These chunks, collectively referred to as the prologue segment, are no different from chunks in other segments. They can be chunks that define objects, or they can be chunks that define other information such as gAMA, cHRM, and sPLT. If any chunks appear between the SAVE chunk and the first SEEK chunk, these chunks also form a part of the prologue segment, but their contents become undefined when the SEEK chunk appears.

Only one instance of the SAVE chunk is permitted in an AniPNG datastream. It is not allowed anywhere after the first SEEK chunk.

It is not permitted, at any point beyond the SAVE chunk, to modify or discard any object that was defined ahead of the SAVE chunk.

An object appearing ahead of the SAVE chunk 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 object buffer that the original object points to.

A chunk like gAMA that overwrites a single current value is permitted after the SAVE chunk, even if the chunk has appeared ahead of the SAVE chunk. Decoders are responsible for saving a copy of the chunk data (in any convenient form) when the SAVE chunk is encountered and restoring it when skipping or jumping to a SEEK chunk from the interior of a segment. If no instance of the chunk appeared ahead of the SAVE chunk, the decoder must restore the chunk data to its original "unknown" condition when it skips or jumps to a SEEK chunk from the interior of a segment.

It is the *encoder's* responsibility, if it changes or discards any "saved" data, to restore it to its "saved" condition (or to nullify it, if it was unknown) prior to the end of the segment. This makes it safe for simple decoders to ignore the SAVE/SEEK mechanism.

Known chunks in this category include DEFI, FRAM, BACK, PLTE, cHRM, tRNS, gAMA, iCCP, bKGD, sBIT, pHYg, pHYs, and sRGB. In addition, it is the responsibility of the encoder to include chunks that restore the potential visibility, location, and clipping boundaries of any "frozen" objects to their "saved" condition.

In the case of chunks like sPLT that can occur multiple times, with different "purpose" fields, additional instances of the chunk are permitted after the SAVE chunk, but not with the same keyword as any instances that occurred ahead of the SAVE chunk. The decoder is required to forget such additional instances when it skips or jumps to a SEEK chunk from the interior of a segment, but it must retain those instances that were defined prior to the SAVE chunk. Encoders are required to nullify such additional instances prior to the end of the segment. Known chunks in this category include only sPLT.

If an entry for a segment (entry type 0 or 1) appears in the optional index, there must also be an entry for every segment, whether named or not, except for the prologue segment, that precedes it. All entries must appear in the index in the same order that they appear in the AniPNG datastream. There must never be a segment entry (type 0 or 1) for the prologue segment, but there can be entries for named images or subframes in the prologue, placed ahead of the first segment entry. Only named images or subframes are permitted, and it is not an error to omit any or all named images or subframes. Nor is it an error to omit a contiguous set of segments at the end of the datastream from the index.

Offsets are calculated from the first byte of the AniPNG 8-byte signature, which has offset=0. This is true even if the AniPNG datastream happens to be embedded in some other file and the signature bytes are not actually present.

Applications with direct access to the datastream can use the index to find segments, subframes, and exported images quickly. After processing the prologue segment, they can jump directly to any segment and then process the remaining datastream until the desired subframe, image, or time is found. Applications that have only streaming access to the datastream can still use the index to decide whether to decode the chunks in a segment or to skip over them.

Only one instance of the SAVE chunk is permitted in an AniPNG datastream. If the SEEK chunk is present, the SAVE chunk must be present, prior to the first SEEK chunk. The only chunks not allowed ahead of the SAVE chunk are the SEEK chunk and the MEND chunk. The SAVE chunk must not appear inside a LOOP-ENDL pair.

4.4.2 SEEK Seek point

The SEEK chunk marks positions ("seek points") in the AniPNG datastream where a restart is possible, and where the decoder must restore certain information to the condition that existed when the SAVE chunk was processed, if it has skipped or jumped to the SEEK chunk from the interior of a segment.

The SEEK chunk can be empty, or it can contain a segment name.

```
Segment_name: 1-79 bytes (Latin-1 string).
```

The segment name is optional. It 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. There is no null byte terminator within the segment name, nor is there a separate null byte terminator. Segment names are case-sensitive. Use caution when printing or displaying keywords (Refer to Security considerations, below, Chapter 14). No specific use for the segment name is specified in this document, but applications can use the segment name for such purposes as constructing a menu of seek points for a slide-show viewer. It can be included in the optional index that can appear in the SAVE chunk. It is recommended that the same name not appear in any other SEEK chunk or in any FRAM or eXPI chunk. Segment names should not begin with the case-insensitive strings “CLOCK(”, “FRAME(”, or “FRAMES(”, which are reserved for use in URI queries and fragments (see Uniform Resource Identifier below).

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

- Data appearing in the MHDR and TERM chunks.
- Anything appearing ahead of the SAVE chunk.

They also must not depend on anything that has been drawn on the output buffer or device. Its contents become undefined when the SEEK chunk is encountered. Viewers that make random access to a seek point from the interior of a segment must insert a background layer before processing the segment. Encoders must ensure that simple viewers do not need to do this.

When the SEEK chunk is encountered, the decoder can discard any objects appearing after the SAVE 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.

When a decoder jumps to a seek point from the interior of a segment, it must restore the information that it saved when it processed the SAVE chunk, and it must reset the object attributes for object 0 to their default values. When it encounters a SEEK chunk during normal sequential processing of an AniPNG datastream, it need not restore anything, because the encoder will have written chunks that restore all saved information.

Multiple instances of the SEEK chunk are permitted. The SEEK chunk must not appear prior to the SAVE chunk. The SAVE chunk must also be present if the SEEK chunk is present. The SEEK chunk must not appear between a LOOP chunk and its ENDL chunk.

4.5 Ancillary AniPNG chunks

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

4.5.1 eXPI Export image

The eXPI chunk takes a snapshot of a viewable object, 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).
Snapshot_name: 1-79 bytes (Latin-1 text).
```

When the snapshot_id is zero, the snapshot is the first instance of an embedded image with object_id=0 following the eXPI chunk. When the snapshot_id is nonzero, the snapshot is an already-defined object with that object_id as it already exists when the eXPI chunk is encountered.

Note that the `snapshot_name` is associated with the snapshot, not with the `snapshot_id` nor its subsequent contents; changing the image identified by `snapshot_id` will not affect the snapshot. The `snapshot_name` means nothing inside the scope of the AniPNG specification, except that it can be included in the optional index that can appear in the SAVE chunk. 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 or SEEK chunk. A decoder that knows of no "outside world" can simply ignore the eXPI chunk. This chunk could be used in AniPNG 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 AniPNG. Thumbnail images are regular PNG images, but they would normally have smaller dimensions and fewer colors than the AniPNG 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 (see Uniform Resource Identifier below).

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

4.5.2 pHYg Physical pixel size (global)

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

Conceptually, an AniPNG 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 against the frame.

AniPNG datastreams can include both the PNG pHYs chunk (either at the AniPNG top level or within the PNG and JNG datastreams) and the AniPNG pHYg chunk (only at the AniPNG top level), to ensure that the images are properly displayed either when displayed by an AniPNG 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 AniPNG top-level pHYg chunk can be nullified by a subsequent empty pHYg chunk appearing in the AniPNG top level.

4.6 Ancillary PNG chunks

The namespace for AniPNG chunk names is separate from that of PNG. Only those PNG chunks named in this paragraph are also defined at the AniPNG 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. The tIME chunk an 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 AniPNG was modified. In a regular segment (between SEEK chunks or between the final SEEK chunk and the MEND chunk), it indicates the last time that segment was modified.

  An AniPNG 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 AniPNG 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 AniPNG top level.

  In the AniPNG 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.

  An AniPNG 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. 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.

  The top-level sRGB chunk nullifies the preceding top-level gAMA and cHRM chunks, if any, and either the top-level gAMA or the top-level cHRM chunk nullifies the preceding top-level sRGB chunk, if any.

- sPLT

  This PNG chunk is also defined at the AniPNG top level. It provides a value that takes precedence over those that might be provided in subsequent PNG or JNG datastreams and provides a value to be used when it is not provided in subsequent PNG or JNG datastreams. It also takes precedence over the PLTE chunk in a subsequent PNG datastream when the PLTE and hIST chunks are 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 an AniPNG datastream. If a `palette_name` is repeated, the previous palette having the same `palette_name` is replaced. It is not permitted, at the AniPNG top level, to redefine a palette after the SAVE chunk with the same `palette_name` as one that appears ahead of the SAVE chunk. It is permitted, however, to define and redefine other palettes with other `palette_name` fields. A single empty sPLT chunk can be used to nullify all sPLT chunks that have been previously defined in the AniPNG top level, except for those that appeared ahead of the SAVE chunk, when the SAVE chunk has been read.

When a decoder needs to choose between a suggested palette defined at the AniPNG 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 precedence to the palette from the AniPNG level, to avoid spurious layer-to-layer color changes.

AniPNG editors that write PNG datastreams should ignore the sPLT data from the AniPNG level and simply copy any sPLT chunks appearing within the embedded PNG datastreams.

5 Extension and Registration

New public chunk types, and additional options in existing public chunks, can be proposed for inclusion in this specification by contacting the PNG/AniPNG specification maintainers at png-mng-misc at lists.sf.net, or png-group at w3.org.

New public chunks and options will be registered only if they are of use to others and do not violate the design philosophy of PNG and AniPNG. Chunk registration is not automatic, although it is the intent of the authors that it be straightforward when a new chunk of potentially wide application is needed. Note that the creation of new critical chunk types is discouraged unless absolutely necessary.

Applications can also use private chunk types to carry data that is not of interest to other applications.

Decoders must be prepared to encounter unrecognized public or private chunk type codes. If the unrecognized chunk is critical, then decoders should abandon the segment, and if it is ancillary they should simply ignore the chunk. Editors must handle them as described in the following section, Chunk Copying Rules.

6 Chunk Copying Rules

The chunk copying rules for AniPNG are the same as those in PNG, except that an AniPNG editor is not permitted to move unknown chunks across any of the following chunks, or across any critical chunk in a future version of this specification that creates or displays an image:

- SAVE
- SEEK
- IHDR
- JHDR

- IEND
- PlAY
- ReCO

The copy-safe status of an unknown chunk is determined from the chunk name, just as in PNG. If bit 5 of the first byte of the name is 0 (Normally corresponding to an uppercase ASCII letter), the unknown chunk is critical and cannot be processed or copied. If it is 1 (usually corresponding to a lowercase ASCII letter), the unknown chunk is ancillary and its copy-safe status is determined by bit 5 of the fourth byte of the name, 0 meaning copy-unsafe and 1 meaning copy-safe.

If an editor makes changes to the AniPNG datastream that render unknown chunks unsafe-to-copy, this does not affect the copy-safe status of any chunks beyond the next SEEK chunk or prior to the previous one. However, if it makes such changes prior the SAVE chunk, this affects the copy-safe status of all top-level unknown chunks in the entire AniPNG datastream.

Changes to the MHDR chunk do not affect the copy-safe status of any other chunk.

The SAVE, SEEK, and TERM chunks are not considered to be a part of any segment. Changes to the data in the SAVE or SEEK chunks do not affect the copy-safe status of any other chunks. Adding or removing a SEEK chunk affects the copy-safe status of unknown chunks in the newly-merged or newly-separated segments. Adding, removing, or changing the TERM chunk has no effect on the copy-safe status of any chunk.

As in PNG, unsafe-to-copy ancillary chunks in the top-level AniPNG datastream can have ordering rules only with respect to critical chunks. Safe-to-copy ancillary chunks in the top-level AniPNG datastream can have ordering rules only with respect to the SAVE, SEEK, PlAY, IHDR-IEND, JHDR-IEND sequences, or with respect to any other critical “header-end” sequence that might be defined in the future that could contain IDAT or similar chunks.

The copying rules for unknown chunks inside IHDR-IEND, and JHDR-IEND sequences are governed by the PNG and JNG specifications, and any changes inside such sequences have no effect on the copy-safe status of any top-level AniPNG chunks.

7 Minimum Requirements for AniPNG-Compliant Decoders

This section specifies the minimum level of support that is expected of AniPNG-compliant decoders, and provides recommendations 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.

Anything less than this level of support requires subsetting.

We are allowing conformant decoders to skip twelve-bit JNGs because those are likely to be rarely encountered and used only for special purposes.

7.1 Required AniPNG chunk support

MHDR

The `ticks_per_second` must be supported by animation viewers. The frame count, layer count, and nominal play time can be ignored.

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.

LOOP, ENDL

The `iteration_count` must be supported. The `nest_level` should be used as a sanity check but is not required. When `iteration_min=1` either explicitly or when it is omitted and the `termination_condition` is not 0 or 4, the LOOP chunk and its ENDL chunk can be ignored.

DEFI

Must be fully supported.

All objects can be treated as “abstract”.

BACK, DISC

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.

SAVE and SEEK

Must be recognized but can be ignored. However, partial support is recommended: All existing objects should be marked “frozen” when the SAVE chunk is processed, so that unneeded objects can be discarded when the SEEK chunk or an empty DISC chunk is processed. The SEEK chunk should be processed as if it were an empty DISC chunk, as a minimum. Chunk information need only be “saved” and “restored” when the viewer is able to skip or jump to random SEEK chunk locations from the interior of a segment, such as when recovering from a corrupted datastream or from a segment containing an unknown critical chunk, or when escaping from a deterministic loop in response to a user request. The optional index can be ignored. Slide-show controllers may wish to support SAVE and SEEK fully.

TERM

Must be recognized but can be ignored.

7.2 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.

tRNS

The PNG tRNS chunk, although it is an ancillary chunk, must be supported in AniPNG-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.

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. If they support the gAMA chunk, they must also support the sRGB chunk, at least to the extent of interpreting it as if it were a gAMA chunk with gamma value 0.45455.

7.3 Required JNG chunk support

AniPNG-compliant decoders must support JNG.

JHDR, JDAT, IDAT, JDAA, 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.

8 Recommendations for Encoders

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

8.1 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 AniPNG, using 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 AniPNG datastream, decoders can improve the speed and consistency of the display.

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

8.2 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.

8.3 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
```

8.4 Embedded images in LOOPs

Embedded images should not be enclosed in loops unless absolutely necessary. It is better to store them ahead of time and then use PlAY chunks inside the loops.

8.5 Tall vs wide layouts of objects used in playlists

Typically the best compression is obtained when a PNG image is very wide and not very tall, with similar tiles adjacent.

8.6 Including optional index in SAVE chunk

Authors of AniPNG files that are intended for transmission over a network should consider whether it is more economical for the client to rebuild the index from scratch than it is to transmit it. Web pages that are likely to be downloaded over slow lines, and whose clients are unlikely to use the index anyway, generally should have empty SAVE chunks. No information is lost by deleting the index, because the AniPNG datastream contains all of the information needed to build the index. If an application does build an index, and the file is going to be kept as a local file, the application should replace the empty SAVE chunk with one containing the index. See above (Paragraph 4.4.1).

8.7 Interleaving JDAT, JDAA, and IDAT chunks

When a JNG datastream contains an alpha channel, and the file is intended for transmission over a network, it is useful to interleave the IDAT or JDAA and the JDAT chunks. In the case of sequential JPEG, the interleaving should be arranged so that the alpha data arrives more or less in sync with the color data for the scanlines. In the case of progressive JPEG, the alpha data should be interleaved with the first JPEG pass, so that *all* of the alpha data has arrived before the beginning of the second JPEG pass.

8.8 Use of the JDAA chunk

It is recommended that the JDAA chunk be used only to convey smoothly varying alpha channels and not to convey binary transparency which is more precisely and efficiently conveyed in IDAT chunks.

9 Recommendations for Decoders

9.1 ENDL without matching LOOP

If a decoder reads an ENDL chunk for which the matching LOOP chunk is missing, or has been skipped for some reason, any active loops with a higher `nest_level` should be terminated, and processing can resume after the next SEEK chunk. Simple viewers that do not process the SAVE chunk should abandon the AniPNG datastream. See above.

9.2 Note on compositing

The PNG specification gives a good explanation of how to composite a partially transparent image over an opaque image, but things get more complicated when both images are partially transparent.

Pixels in PNG and JNG images are represented using gamma-encoded RGB (or gray) samples along with a linear alpha value. Alpha processing can only be performed on linear samples. This chapter assumes that R, G, B, and A values have all been converted to real numbers in the range [0..1], and that any gamma encoding has been undone.

For a top pixel {Rt,Gt,Bt,At} and a bottom pixel {Rb,Gb,Bb,Ab}, the composite pixel {Rc,Gc,Bc,Ac} is given by:

```
Ac = 1 - (1 - At)(1 - Ab)
if (Ac != 0) then
  s = At / Ac
  t = (1 - At) Ab / Ac
else
  s = 0.0
  t = 1.0
endif
Rc = s Rt + t Rb
Gc = s Gt + t Gb
Bc = s Bt + t Bb
```

When the bottom pixel is fully opaque (Ab = 1.0), the function reduces to:

```
Ac = 1
Rc = At Rt + (1 - At) Rb
Gc = At Gt + (1 - At) Gb
Bc = At Bt + (1 - At) Bb
```

When the bottom pixel is not fully opaque, the function is much simpler if premultiplied alpha is used. A pixel that uses non-premultiplied alpha can be converted to premultiplied alpha by multiplying R, G, and B by A.

For a premultiplied top pixel {Rt,Gt,Bt,At} and a premultiplied bottom pixel {Rb,Gb,Bb,Ab}, the premultiplied composite pixel {Rc,Gc,Bc,Ac} is given by:

```
Ac = 1 - (1 - At)(1 - Ab)
Rc = Rt + (1 - At) Rb
Gc = Gt + (1 - At) Gb
Bc = Bt + (1 - At) Bb
```

As mentioned in the PNG specification, the equations become much simpler when no pixel has an alpha value other than 0.0 or 1.0, and the RGB samples need not be linear in that case.

9.3 Retaining object data

The decoder must retain information about each object (except for objects with `object_id=0`) for possible redisplay with the PlAY chunk.

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

- The set of object attributes (potential visibility, location, clipping boundary data from the DEFI chunks, and the pointer to an object buffer).
- The pixel data and the values associated with other recognized PNG chunks such as PLTE and gAMA, subject to the chunk copying rules. It may be stored in any convenient form.

When the encoder knows that data in the object buffer will not be needed later, it can help decoders operate more efficiently by using `object_id=0` or by using the DISC or the SEEK chunk.

9.4 Decoder handling of fatal errors

When a fatal error is encountered, such as a bad CRC or an unknown critical AniPNG chunk, minimal viewers that do not implement the SAVE/SEEK mechanism should simply abandon the AniPNG datastream.

More capable AniPNG viewers should attempt to recover gracefully by abandoning processing of the segment and searching for a SEEK chunk. If such errors occur before the SAVE chunk is reached, the viewer should abandon the AniPNG datastream.

When an error occurs within a image datastream, such as an unknown critical PNG chunk or a missing parent object where one was required, only that image should be abandoned and the associated object should be discarded. If a bad CRC is found, indicating a corrupted datastream, the entire segment should be abandoned, as above.

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

9.5 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 AniPNG authors should not assume that the frames will be displayed in other than their final form.

9.6 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. Encoders can supply a top-level sPLT chunk with a suggested reduced global palette to help decoders build an appropriate palette when necessary.

9.7 Behavior of single-frame viewers

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

9.8 Clipping

AniPNG provides four 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 AniPNG chunk.

Decoders can use these parameters to establish the size of a window in which to display the AniPNG 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 an AniPNG datastream that consists of only 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.

Layer clipping boundaries

The layer 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 the foreground layers of each frame, clear the area within the layer clipping boundaries to the background color, and display the background image if one has been defined, thus creating a separate layer at the beginning of each frame. Viewers must not change any pixels outside the layer boundaries; encoders must be able to rely on the fact that the part of the display that is outside the layer 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 and can be changed by the PlAY chunk. They are associated with individual objects, not with the layers, and they can be changed within a seqence of layers. They are useful for exposing only a portion of an image in a frame, to achieve effects such as scrolling, panning, or gradual exposure.

The clipping boundaries are expressed in pixels, measured rightward and downward from the frame origin.

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 the pixel falls within the physical limits of the display hardware and all of the following are true:

```
0         <= x < frame_width  (from the MHDR chunk)
0         <= y < frame_height
Left_lcb <= x < right_lcb    (from the FRAM chunk)
Top_lcb  <= y < bottom_lcb
Left_cb  <= x < right_cb     (from the DEFI chunk)
Top_cb   <= y < bottom_cb
```

PlAY clipping boundaries

One type of clipping performed in PlAY gives a fourth type that has no dependencies on the other types, since the object clipping boundaries are ignored and the PlAY chunk defines its own clipping boundaries with respect to the upper left corner of the source object. The left and top of this type of clipping is also inclusive, and the right and bottom are exclusive.

10 Recommendations for Editors

10.1 Editing datastreams with optional index

Editors must recreate or delete the optional SAVE chunk index whenever they make any change that affects the offsets of chunks following the portion of the datastream that is changed. If the changes do not involve the addition, deletion, or relocation of segments, frames, and images, then it is sufficient to zero out the offsets.

The SAVE chunk is not considered to be in any AniPNG segment, so changing it has no effect on the copy-safe status of unknown chunks in any other part of the AniPNG datastream.

When the SAVE chunk is expanded to include an index, all chunks that follow will have their offsets changed by an amount equal to the change in the length of the data segment of the SAVE chunk, so the offset table will have to be adjusted accordingly. If a SAVE chunk is already present with zero offsets, the correct offsets can be written without adjustment.

10.2 Handling LOOP and TERM chunks

Editors that create a series of PNG or JNG datastreams from an AniPNG datastream should check the termination condition of any LOOP chunks and execute loops only `iteration_min` times. The loop created by the TERM chunk should be executed only once.

11 Miscellaneous Topics

11.1 File name extension

On systems where file names customarily include an extension signifying file type, the extension `.mng` is recommended for AniPNG files. Lowercase `.mng` is preferred if file names are case-sensitive.

11.2 Internet media type

When and if the AniPNG format becomes finalized, the AniPNG authors intend to register `video/mng` as the Internet Media Type for AniPNG [RFC-2045], [RFC-2048]. This is the same media type that was intended for the MNG-1.0 format but never registered.

At the date of this document, the media type registration process had not been started. It is recommended that implementations also recognize the interim media type `video/x-mng`.

11.3 Uniform Resource Identifier (URI)

Segments, subframes, and objects are externally accessible via named SEEK, eXPI, and FRAM chunk names. They can be referred to by URI, as in

```
SRC=file.mng#segment_name
SRC=file.mng#subframe_name
SRC=file.mng#snapshot_name
SRC=file.mng?segment_name#segment_name
SRC=file.mng?snapshot_name#snapshot_name
```

When the URI specializer ("#" or "?") is "#", and the fragment identifier (the string following the specializer) is the name of a segment, i.e., a named SEEK chunk, the viewer should display the sequence from the beginning of the named segment up to the next segment. When it refers to a subframe or an image, i.e., a named FRAM or eXPI chunk, it should display the single frame (as it exists when the next FRAM chunk is encountered) or image that is identified by the fragment identifier. The client can find the needed segment quickly if the SAVE chunk is present and contains the optional index.

When the URI specializer is "?" (server side query), the "query component" is the string following the "?" specializer and up to but not including the "#" if the "#" specializer is also present. The server should find the segment that is named in the query component or the segment that contains the subframe or image named in the query component, and it should return a datastream consisting of:

- all chunks prior to the SAVE chunk,
- an empty SAVE chunk,
- the SEEK chunk for the segment being returned,
- all chunks in the segment, and
- a MEND chunk.

If no SAVE chunk is present, the server must simply return the entire AniPNG datastream. Servers that are unwilling to parse the AniPNG datastream and are unconcerned about bandwidth can return the entire AniPNG datastream even when the SAVE chunk is present. Authors should defend against this behavior by including both a query and a fragment in the URI even when a segment is being requested.

The client can process this as a complete AniPNG datastream, either displaying the entire segment, if no fragment identifier is present, or extracting the segment, frame, or image that is named in a fragment identifier and displaying it, if a fragment identifier is present (a fragment identifier must be present if a frame or image is being requested). To "extract a frame" means to decode the returned datastream through the end of the frame that contains the named subframe and to display the result as a single still image. If the layers of the named subframe do not cover the entire frame, pixels from the background and from earlier subframes must be included in the resulting composition.

A part of the AniPNG datastream can also be requested by timecode, as in

```
SRC=file.mng#clock(10s-20s)
SRC=file.mng#clock(0:00-0:15)
SRC=file.mng?clock(0:00-0:15)#clock(0:00-0:15)
```

or by frame number, as in

```
SRC=file.mng#frame(10)
SRC=file.mng#frames(30-60)
SRC=file.mng?frames(30-60)#frames(30-60)
```

The timecode must consist of starting and ending clock values, as defined in the W3C SMIL recommendation, separated by a hyphen (ASCII code 45).

When the URI specializer is "#", the viewer should play that part of the sequence beginning and ending at the requested times, measuring from zero time at the beginning of the AniPNG datastream, or beginning and ending with the specified frame numbers. To do this it must start with the segment containing the requested time and decode any part of the segment up to that time, composing but not displaying the frames. This will provide the background against which the desired frames are displayed.

When the URI specializer is "?", the server can send the entire AniPNG datastream, or, preferably, it should construct a complete AniPNG file containing:

- the chunks preceding the SAVE chunk,
- the SAVE chunk itself with an optional index that gives the starting time and starting frame number of the first SEEK chunk that is sent, and
- the one or more consecutive sets of segments, with their SEEK chunks, that contain the sequence beginning and ending at the requested times, or frame numbers, at the proper framing rate.

If the server does not send the entire AniPNG datastream, and the first segment after the SAVE chunk is not sent but a later segment *is* sent, the optional index must be written even if it does not exist in the source file. The index must contain at least one "type 0" entry that gives the nominal start time and frame number for the first segment that is sent after the SAVE chunk. The offset field can be set to zero and the segment name can be omitted.

The query component should always be repeated as a fragment identifier, so clients can find the requested item in case the server sends more than what was requested.

AniPNG datastreams should not contain segment, subframe, or image names that begin with the case-insensitive strings "CLOCK(", "FRAME(", or "FRAMES(", which are reserved for use in URI queries and fragments (see Uniform Resource Identifier below).

See [RFC-2396] and the W3C SMIL recommendation at http://www.w3.org/TR/.

12 Revision History

12.1 Version 0.7

Released 28 September 2007

- Allow object 0 to be a source for the PlAY chunk playlist.
- Made coordinate system for PlAY consistent with other chunks.
- Removed abstract/concrete discussion. Added suggested simplicity profile.
- Use unregistered ReCO and PlAY chunk names instead of RECO and PLAY¡/cn)
- Trivial editorial changes.

12.2 Version 0.1

Released 9 September 2007

- Initial release, mostly extracted from the MNG-1.0 specification.
- Eliminated the simplicity profile, MNG-VLC, and MNG-LC subsets.

- Combined the MOVE, CLIP, PAST, and SHOW chunks into the PLAY and RECO chunks.
- Eliminated Delta-PNG and all of its chunks: DHDR, PROM, IPNG, PPLT, IJNG, DROP, DBYK, and ORDR.
- Eliminated the BASI and CLON chunks, which are of little use without Delta-PNG.
- Eliminated the MAGN chunk. It had a design flaw, in that it did not require interpolation to be done with linear color samples.
- Eliminated the seldom-used fPRI and nEED chunks.
- Allow standalone PNG and JNG files to bear the AniPNG signature.
- Removed the JNG specification from this document.

13 References

[ISO/IEC-10918-1]
International Organization for Standardization and International Electrotechnical Commission, "Digital Compression and Coding of Continuous-tone Still Images, Part 1: Requirements and guidelines," ISO/IEC IS 10918-1, ITU-T T.81.
See also Pennebaker, William B., and Joan L. Mitchell, "JPEG: Still Image Data Compression Standard," Van Nostrand Reinhold, ISBN:0442012721, September 1992

[JFIF]
C-Cube Microsystems, "JPEG File Interchange Format, Version 1.02," September 1992.

[JNG]
Randers-Pehrson, G., et al, "JNG (JPEG Network Graphics) Format,"
ftp://ftp.simplesystems.org/pub/png/documents/.

[LOCO]
Weinberger, Marcelo J., Gadiel Seroussi, and Guillermo Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS" Hewlett Packard Report HPL-98-193R1, November 1998, revised October 1999,, available at http://www.hpl.hp.com/loco/.

[MNG]
Randers-Pehrson, G., et al, "MNG (Multiple-image Network Graphics Format,"
ftp://ftp.simplesystems.org/pub/png/documents*.

[PNG]
Boutell, T., et. al., "PNG (Portable Network Graphics Format) Version 1.0," RFC 2083,
ftp://ftp.isi.edu/in-notes/rfc2083.txt also available at
ftp://ftp.simplesystems.org/pub/png/documents/. This specification has also been published as a W3C Recommendation, which is available at
http://www.w3.org/TR/REC-png.html.

See also the PNG-1.2 specification:
Randers-Pehrson, G., et. al., "PNG (Portable Network Graphics Format) Version 1.2," which is available at
ftp://ftp.simplesystems.org/pub/png/documents/.

[PNG-EXT]
Randers-Pehrson, G., et al, "Extensions to the PNG 1.2 Specification,"
ftp://ftp.simplesystems.org/pub/png/documents/pngext-*.

[RFC-2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119/BCP 14, Harvard University, March 1997.

[RFC-2045]
Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC 2045, Innosoft, First Virtual, November 1996.
ftp://ftp.isi.edu/in-notes/rfc2045.txt

[RFC-2048]
Freed, N., Klensin, J., and J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures," RFC 2048, Innosoft, MCI, USC/Information Sciences Institute, November,nbsp;1996.
ftp://ftp.isi.edu/in-notes/rfc2048.txt

[RFC-2396]
Berners-Lee, T., R. Fielding, U. C. Irvine, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, Xerox Corporation, University of Minnesota, August 1998.
ftp://ftp.isi.edu/in-notes/rfc2396.txt

14 Security Considerations

Security considerations are addressed in the PNG specification.

An infinite or just overly long loop could give the appearance of having locked up the machine, as could an unreasonably long interframe delay or a misplaced `sync_id` with a long `timeout` value. Therefore a decoder should always provide a simple method for users to escape out of a loop or delay, either by abandoning the MNG entirely or just proceeding to the next SEEK chunk. Decoders should check for user input after each loop iteration (not just after each frame) in case of infinite loops that are empty or that generate layers with zero interframe delay. The SEEK chunk makes it safe for a viewer to resume processing after it encounters a corrupted portion of a MNG datastream or jumps out of the interior of a segment for any reason.

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.

15 Credits

Acknowledgments

The following persons contributed to the development of the PlAY chunk or its predecessor, the proposed PNG aNIM chunk:

- Adeluc, png at adeluc.com
- John Bowler, jbowler at acm.org
- Gerard Juyn, gjuyn at xs4all.nl
- Chris Lilley, chris at w3.org
- The authors of the PNG and MNG specifications
- Contributors to the png-mng-misc at lists.sourceforge.net list

Thanks to the following person for carefully proofreading several versions of this document:

- Nancy M. Randers-Pehrson, nancyrp at comcast.net

Trademarks

- GIF is a service mark of CompuServe Incorporated.
- X Window System is a trademark of the Massachusetts Institute of Technology.

Document source

This document was built from the file `anipng-0.7-20070928.html` on 28 September 2007.

Copyright Notice

Copyright © 2007 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 HOLDER MAKES 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 AniPNG Specification.