

Overview

The Animated Portable Network Graphic (APNG) file format is similar to an animated GIF. The Portable Network Graphic Animator (PNGA) software can construct an APNG file from a collection of PNG image files and XML meta data. The XML entries must be provided by the user. The meta data will be combined with the PNG image data to construct an APNG file that can be viewed using the Mozilla Firefox web browser amongst other software. In addition, the Portable Network Graphic Animator software can accept an existing APNG file and convert its frames into individual PNG image files, along with XML data with the information for each frame. Overall, this allows creation of new APNG animations as well as modification of existing APNG animations.

PNGA XML Schema

PNGA uses a simple XML file to describe an animated PNG. Everything is a child of the <pnga> tag. Its attributes frames and plays are used to indicate the number of frames in the animation and the number of times the animation is looped respectively. When plays is set to 0 the animation will loop forever. The list of <image> tags are used to indicate each frame in the animation in order. Its attributes src and length are the png to use for the current frame and the number of seconds to display each frame respectively. Length is usually specified as a rational value with something like “1/25” meaning display this frame for 1/25th of a second. Here is an example PNGA XML file contents:

```
<pnga frames="5" plays="2">
  <image src="bob0.png" length="1/12"/>
  <image src="bob1.png" length="1/14"/>
  <image src="bob2.png" length="1/14"/>
  <image src="bob3.png" length="1/14"/>
  <image src="bob4.png" length="1/10"/>
</pnga>
```

Modules

IO

- **animate (xmlfilename, state)**
Writes an animated portable network graphic file to disk.
xmlfilename = (string) the name of the XML document containing information on number of frames, number of plays, PNG filenames, and the length of time each frame is displayed. The output is an animated PNG file with the same base filename and a .pnga extension.
- **suspend (apngfilename, state)**
Writes a series of portable network graphic files to disk, along with an XML document. These image files are the individual frames of the APNG file in the parameter, and the XML document contains information on the number of frames, number of plays, PNG filenames, and length of time each frame is displayed.
apngfilename = (string) the name of the animated portable network graphic to be broken into individual frames. This will be used as the base file name for the PNGs and the XML document. For PNG frames, it will be suffixed by the frame number. E.g., if apngfilename is "ball", the XML file will be written to ball.xml, and the frames will be written to ball1.png, ball2.png, et cetera.
- **writeFiles (filelist, state)**
Helper function for suspend. Writes PNG files to disk.
filelist = list of list (filename filedata).
- **openFile (frame, state)**
Helper function for animate, calls binary-file->byte-list on a filename frame
frame = filename of the png frame to open
- **openFiles (framelist, state)**
Helper function for animate, calls openFile on the framelist returning the byte-lists of each function
framelist = list of frame file names
- **nameTheseFrames (frameData apngFileName)**
Helper function for writeFrames
Delivers a list of generic file names for framedata along with the time lengths. The order of the files are reversed.
FrameData = APNG file data in the form: ((framedata1 time_for_frame1) (framedata2 time_for_frame2))...
apngFileName = name of the apng file

XMLUtils

- **parseXML (domXML)**
Parses XML data and delivers the number of frames, number of plays, and a list of filenames with their corresponding time length.
domXML = XML data as a document object model
- **grabFrames (framelist)**
Delivers a list, where each item corresponds to the metadata for each image element specified in the xml config. An item consists of two elements: the file name and the length of time the frame should be displayed.
framelist = list of minidom "image" nodes
- **numplays (processedConfigDOM)**
Helper function given processedConfigDOM returns the numplays
processedConfigDOM = the config structure returned by processXML
- **numframes (processedConfigDOM)**
Helper function given processedConfigDOM returns numframes
processedConfigDOM = the config structure returned by processXML
- **framelist (processedConfigDOM)**
Helper function given processedConfigDOM returns the framelist
processedConfigDOM = the config structure returned by processXML
- **writeXML (numPlays, numFrames, framedata)**
Delivers a string that is an XML document containing the information for an APNG file.
numPlays = the number of times the animation will play.
numFrames = the total number of frames that make up the animation.
framedata = a list of list (PNGfilename, corresponding time length).
- **nameFrames (frameData apngFilename)**
Helper function for writeFrames
Delivers a list of generic file names for framedata along with the time lengths. The order of the files are reversed.
FrameData = APNG file data in the form: ((framedata1 time_for_frame1)
(framedata2 time_for_frame2)...
apngFileName = name of the apng file
- **writeFrames (frameData apngFilename)**
Delivers a list of filenames with their time lengths for APNG frame data.
frameData = APNG file data in the form: ((framedata1 time_for_frame1)
(framedata2 time_for_frame2)...
apngFileName = name of the apng file

BasicLex – 154 Base LOC

- split-at-delimiter (ds xs)** → ds = delimiters to look for (list)
 xs = object to search in (list)
 (split-at-delimiter ds xs) = (before at+)
 where before = longest prefix of xs with no values from ds (list)
 at+ = rest of xs (list)
- span (ps xs)** → ps = list of signals to pass by (no constraints on signals)
 xs = list of signals (no constraints on signals)
 (span ps xs) = list of two elements
 1 longest prefix of xs containing only signals from ps
 2 rest of xs
- splitoff-prefix (ps xs)** → ps = prefix to look for (list)
 xs = object to search in (list)
 (splitoff-prefix ps xs) = (ps-matching ps-af-match xs-af-match)
 where ps-matching = longest ps prefix matching xs prefix (list)
 ps-af-match = rest of ps (list)
 xs-af-match = non-matching suffix of xs (list)
 Note: ps-af-match = nil means ps is a prefix of xs
- splitoff-prefix-upr (ps xs)** → ps = prefix to look for (list of standard, upper-case characters)
 xs = object to search in (list)
 (splitoff-prefix-upr ps xs) = (ps-matching ps-af-match xs-af-match)
 where ps-matching = longest ps prefix matching xs prefix (list)
 ps-af-match = rest of ps (list)
 xs-af-match = non-matching suffix of xs (list)
 Notes: 1. search is not sensitive to case of letters in xs
 2. ps-af-match = nil means ps is a prefix of xs
 Implementation issue: combining general and case-insensitive search
 in one function simplifies maintenance, but complicates
 formulation of software properties and their proofs
- splitoff-prefix-chr (tok-str xs)** → tok-str = characters to look for (string, standard characters)
 chrs = object to search in
- split-on-token-gen (tok xs)** → tok = object to search for (list)
 xs = object of search (list)
 (split-on-token-gen tok xs) = (prefix match suffix)
 where
 prefix = elems of xs before 1st sublist matching tok (list)
 = xs if no match
 match = tok if match (list)
 = nil if no match
 suffix = elems of xs after 1st sublist matching tok (list)

= nil if no match

- **split-on-token-chr (tok xs)** → *tok = object to search for (list of upper-case standard characters)*
xs = object to search in (list containing no non-standard chars)
Note: matching is not case-sensitive
(split-on-token-chr tok xs) = (prefix match suffix)
where
prefix = elems of xs before 1st sublist matching tok (list)
= xs if no match
match = tok if match (list)
= nil if no match
suffix = elems of xs after 1st sublist matching tok (list)
= nil if no match
- **split-on-token (tok xs)** → *tok = object to search for (string or list)*
xs = object to search in (list, no non-standard chars if tok is string)
Note: search is not case-sensitive if tok is a string
Warning! Neither tok nor xs may contain non-standard characters
if tok is a string
Implementation issue: combining general and case-insensitive search
in one function simplifies maintenance, but complicates
formulation of software properties and their proofs

MinidomParser – 317 Base LOC

(base from previous project)

- **xml-text (text)** → *returns a text node with specified text.*
- **xml-node (nodetype attributes children)** → *returns a node with specified nodetype attributes and children.*
- **xml-attribute (attrname attrvalue)** → *returns an attribute with specified name and value.*
- **xml-getnodetype (node)** → *return the type of the node*
- **xml-getattrlist (node)** → *return the list of attributes of the node*
- **xml-getchildren (node)** → *return the children of the nodes*
- **xml-getattrname (attribute)** → *return the name of the attribute*

- **xml-getattrvalue (attribute)** → *return the value of the attribute*
- **xml-getnodes (node nodename)** → *returns children of node with type nodename*
- **xml-getdeepnodes (node nodename)** → *returns children of node with type nodename searching recursively using DFS with node as root.*
- **xml-getnode (node nodename)** → *returns first child node with type nodename*
- **xml-getdeepnode (node nodename)** → *returns first child node with type nodename searching recursively using DFS with node as root.*
- **xml-getattribute (node attributename)** → *returns the value of node's attribute with name attributename*
- **xml-gettext (node)** → *returns the composite of all text inside of a node*
- **xml-isattribute (attribute)** → *returns true iff attribute is a list of length 2 with both elements of the list being strings*
- **xml-isattributelist (attributes)** → *returns true iff attributes is nil or a list of attributes*
- **xml-isnode (node)** → *returns true iff node is actually a node*
- **xml-isodelist (nodes)** → *returns true iff nodes is a list of nodes*
- **gluekids (nodes)** → *Given a list of nodes, glue all nodes' children together in one big list; i.e., if the nodes are rooted in some tree where they're at depth k, then take all of the nodes at depth k+1 (cousins or siblings to one another), and put them into a list together.*
- **xml-bfsfindnodes (nodes nodename)** → *returns a list of children nodes of type nodename using BFS search that are at the shallowest depth.*

MinidomSerializer – 125 Base LOC

(base from previous project)

- **xml-readnode (xmlchars)** → *returns the root node from xmlstring*
- **xml-readnodes (xmlchars)** → *returns (list nodes remainingxmlstring)*

- **xml-unescape (escapedchars)** → *string with entities replaced*
- **xml-readnodeproperties (xmlchars)** → *returns (list attributes remainingxmlstring)*
- **xml-skipdontcares (xmlchars)** → *returns next xmlchars sans don't cares*
- **xml-escape (unescapedchars)** → *returns string with bad chars replaced with entities*
- **xml-serizlize-dom (xmlnode)** → *Returns a string containing an xml document that represents the dom passed in through xmlnode.*
- **xml-serialize-attributes (attributes)** → *Returns a string that is xml that represents the passed in attribute list.*
- **xml-serizlize-nodes (xmlnodes)** → *Returns a string containing xml nodes that represents the node list, xmlnodes.*

APNGBuilder

- **buildAPNG (numPlays, numFrames, framedata)**
Returns an APNG as a string that has the following playtime properties:
 - * *contains in order the frames and amount of time to display each frame from framedata*
 - * *loops the frames numPlays times*
 - * *contains the number of frames as specified in numFrames**numPlays = number of times to loop the animation*
numFrames = number of frames in the animation
framedata = frame and time display information of the type (list (list frame displaytime)...)
where frame is a byte-list representing a PNG's contents and displaytime is a string representing the amount of time in seconds to display that frame as a rational number (i.e. 100/2997 for NTSC standard)
- **preparePNG (ihdr, idat, chunks)**
This function formats a list of PNG file chunks into more conveniently utilized chunks, and returns (list IHDR IDAT) where IHDR is the IHDR chunk IDAT is all IDAT chunks concatenated into one chunk.
ihdr = ihdr chunk data, pass in nil initially
idat = idat chunk data, pass in nil initially
chunks = a list of blown chunks
- **preparePNGs (framedata)**
This function formats the raw PNG file data into more conveniently utilized chunks, and returns a list of (list IHDR IDAT displaytime) where IHDR is the IHDR chunk IDAT is the IDAT chunk

*and time is the displaytime is the corresponding value from the framedata parameter.
 framedata = frame and time display information of the type (list (list frame displaytime)...)
 where frame is a byte-list representing a PNG's contents and displaytime is a string
 representing the amount of time in seconds to display that frame as a rational number (i.e.
 100/2997 for NTSC standard)*

- **validateIHDR (prepdPNGs, baseIHDR)**

Scan the list returned from preparePNGs for inconsistencies between PNG files' IHDR and the first frame's IHDR. APNG requires all frames to have the same compression, filter method, and bit depth, and for the width and height to be less than or equal to the first frame. This filter returns true if and only if all frames in prepdPNGs satisfy this property in relation to baseIHDR. If baseIHDR is null then the IHDR is extracted from the first PNG in the list and the function is recalled passing in this IHDR.

*prepdPNGs = prepared list of PNGs in the form described as the output of preparePNGs
 baseIHDR = the reference IHDR or if null the IHDR of the first PNG in the list to which all comparisons of consistency are made.*

- **buildFrames (prepdPNGs, frameNum)**

Convert the prepdPNGs parameter into a byte-list representation of the APNG frames with their associated fcTL, IDAT and fdAT chunks using prepdPNGs as a source for the framedata. This does not include the file signature, IHDR or acTL chunk.

*prepdPNGs = prepared list of PNGs in the form described as the output of preparePNGs
 frameNum = if 0 then the image data of the first PNG of the list will be output as an IDAT chunk all other frames are fdAT chunks*

- **buildACTL (numPlays, numFrames)**

Returns a byte-list representing the acTL chunk described by the parameters.

numPlays = number of times the animation is intended to be played

numFrames = the number of frames in the animation

- **buildFCTL (sequenceNum, width, height, xOffset, yOffset, delayTime, disposeOp, blendOp)**

Returns a byte-list representing the fcTL chunk described by the parameters.

sequenceNum = the sequence number in the animation

width = width of the frame

height = height of the frame

xOffset = xOffset of the frame

yOffset = yOffset of the frame

delayTime = frame delay string in the form of a rational number (i.e. 100/2997) in seconds

disposeOp = after display do 0 = nothing, 1 = transparent black, 2 = revert to previous frame

blendOp = 0 = overwrite all color components including alpha, 1 = blend over

- **verifyPNGSig (framedata)**

Returns a string if the frames in framedata do not have valid PNG sigs.

framedata = frame and time display information of the type (list (list frame displaytime)...)

where frame is a byte list representing a PNG's contents and displaytime is a string

representing the amount of time in seconds to display that frame as a rational number (i.e. 100/2997 for NTSC standard)

PNGUtils

- **bytep (x)**
Returns true if x is a byte value
x = what to check
- **byte-listp (x)**
Returns true if x is a list of byte values
x = what to check
- **chunktypep (x)**
Returns true if x is a 4 character string representing a chunk type
x = what to check
- **chunkp (x)**
Returns true if x is a chunk (list chunktypep byte-listp)
x = what to check
- **chunk-listp (x)**
Returns true if x is a list of chunks
x = what to check
- **list->avl (index, alist)**
Helper function to convert the crc32 lookup table to an avl-tree with the key being it's index-based position in the list.
(deprecated, for reason even nth works faster in this case)
index = the current key
alist = the list of values to be associated with each key
- **crc32LookupCalc (c)**
This helper function can be used to calculate (or precompute) a portion of the crc32.
c = the key for the value to calculate of type bytep
- **crc32Lookup (index)**
Returns the crc32 lookup table value for a given index. This was historically used to lookup a value from a table, but now just maps directly to crc32LookupCalc for speed reasons.
index = the key for the value to lookup of type bytep
- **blowChunks (pngdata)**
After being given a lot of PNG image data, blowChunks processes this data on a chunk by chunk basis and subsequently returns the list of list pairs of chunk type (ascii string) and chunk data (byte list). This function, historically, dropped any chunk with an invalid crc32 but that section of code is commented out for performance reasons.

For example:

```
*PNG Image →      (      (list IHDR ihdr_data) (list IDAT idat_data)
                    ...
                    (list IHDR ihdr_data) (list acTL actl_data)
                    (list fcTL fctl_1)   (list fdAT fdat_1)
                    (list fcTL fctl_2)   (list fdAT fdat_2)
                    ...
                    )
pngdata = raw, unprocessed png data bytes (excluding the PNG signature)
```

- makeChunk (chunktype, chunkdata)**
 Given a chunk type and correctly formatted chunkdata, makeChunk returns the correctly formatted chunk including the chunk length, type, data, and CRC (using calcCRC32).
 chunktype = type of the chunk to be created, a length 4 ascii string
 chunkdata = raw data portion of the chunk to be created as byte list
- makeNum (num, signed, numbytes)**
 Converts the given number into a byte-listp network order representation of the number.
 num = number to be converted
 signed = true means make it two's complement
 numbytes = number of bytes used in representation
- parseNum (bytes, signed, numbytes)**
 Parses a number byte-listp representation of a network order number and returns an ACL2 native number.
 bytes = number to be parsed as a byte-listp
 signed = true means it is two's complement
 numbytes = number of bytes used in representation
- updateCRC32 (crc32 bytes)**
 Given a previously calculated crc32 value and raw data bytes, such as that found in the data portion of a PNG Image chunk, returns an updated CRC value based on the new bytes.
 crc32 = previously computed CRC32
 bytes = raw data from PNG Image or other source
- calcCRC32 (bytes)**
 Given raw data bytes, such as that found in the data portion of a PNG Image chunk, returns the calculated CRC.
 bytes = raw data from PNG Image or other source
- ascii->bytes (string)**
 Turns an ascii string into it's equivalent in bytes.
 string = a string containing only ascii characters
- bytes->ascii (bytes)**
 Turns ascii bytes into it's equivalent string.
 bytes = a list of bytes that represent only ascii characters

APNGExploder

- **explodeAPNG (apngdata) → (numFrames, numPlays (framedata time))**

Given an APNG file, breaks the APNG into its constituent PNG Images. This process involves looking at the acTL chunk to determine number of frames and number of plays, as well as looking at the fcTL and fdAT pairs to reconstruct the IHDR and IDAT chunks of the PNG Images that comprise the APNG input.

Output is as follows:

$$\text{APNG} \rightarrow (\begin{array}{l} \text{numFrames numPlays} \\ \text{(framedata1 time_for_frame1)} \\ \text{(framedata2 time_for_frame2)} \\ \dots \end{array})$$

apngdata = raw apng byte-list given from the IO Module.

- **takeChunk (name, chunklist, checkedchunks)**

Finds the first data chunk in a ((name chunkdata) (name chunkdata)) list that matches the given name and pulls it out of the chunklist. Returns (list chunkdataforname restofchunklist)

name = chunktypep of the chunk to find in the chunklist

chunklist = chunk-listp to search

checkedchunks = a list of chunks to prepend to the list of checked chunks.

- **getChunksWithName (name, chunkstocheck)**

Given a list of chunks iterate through them and build:

a) a list of chunks that match name

b) a a list of chunks that don't match name

The function delivers a two-element list where the first element is the list described in (a) and the second is the list described in (b).

name = the chunktypep of the chunk to find in chunkstocheck

chunkstocheck = the list of chunks to check

- **getIHDR (chunks) → IHDR**

Given APNG chunks, returns the IHDR chunk contained within.

chunks = processed (or raw) data chunks contained within the input APNG

- **getAcTL (chunks) → numFrames numPlays**

Given APNG chunks, returns the numFrames and numPlays contained within the acTL chunk following the IHDR chunk. These two elements completely comprise the acTL chunk.

chunks = processed (or raw) data chunks contained within the input APNG

- **splitAtFirstFrameChunk (IDATflag, pre, post)**

Iterates through post and returns when the head of post has chunktype fcTL, IEND, or IDAT (when IDATflag is true). Returns (list pre post) where pre is a list of chunks traversed and post is the list of chunks starting at the fcTL, IEND, or IDAT chunk.

IDATflag = whether to stop on IDAT chunk or only fcTL and IEND

pre = the list of traversed chunks (probably want to start with nil)

post = the list of chunks to traverse

- **makeChunks (chunklist)**

Makes all the chunks in the list of chunks and concatenates them. This returns a byte-list representing the passed in chunklist

chunklist = list of chunks to convert to their byte list equivalent form

- **splitByFcTL (pre, post)**

Splits the chunklist on the next fcTL. Returns (list pre post)

pre = the list of chunks already traversed (you probably want to pass in nil)

post = the list of chunks to check

- **buildDataChunk (fdat, idat, chunks)**

This function formats a list of PNG chunks into more conveniently utilized chunks, and returns (list fdAT IDAT) where fdAT is all fdAT chunks for this frame concatenated into one chunk and IDAT is all IDAT chunks concatenated into one chunk.

fdat = fdAT chunk data, pass in nil initially

idat = idat chunk data, pass in nil initially

chunks = a list of blown chunks

- **getFrame (chunks, IDATflag, prefix, ihdr)**

Returns a complete PNG file as a byte-list given the following:

chunks = PNG chunks which represent one PNG file from the input APNG

IDATflag = Defines whether or not the IDAT chunks have been pulled yet

prefix = chunks besides IHDR used for entire APNG file, data contained herein will be used when reconstructing every PNG file.

ihdr = the IHDR chunk data as a byte-list

- **getFrames (chunks, IDATflag, prefix, ihdr)**

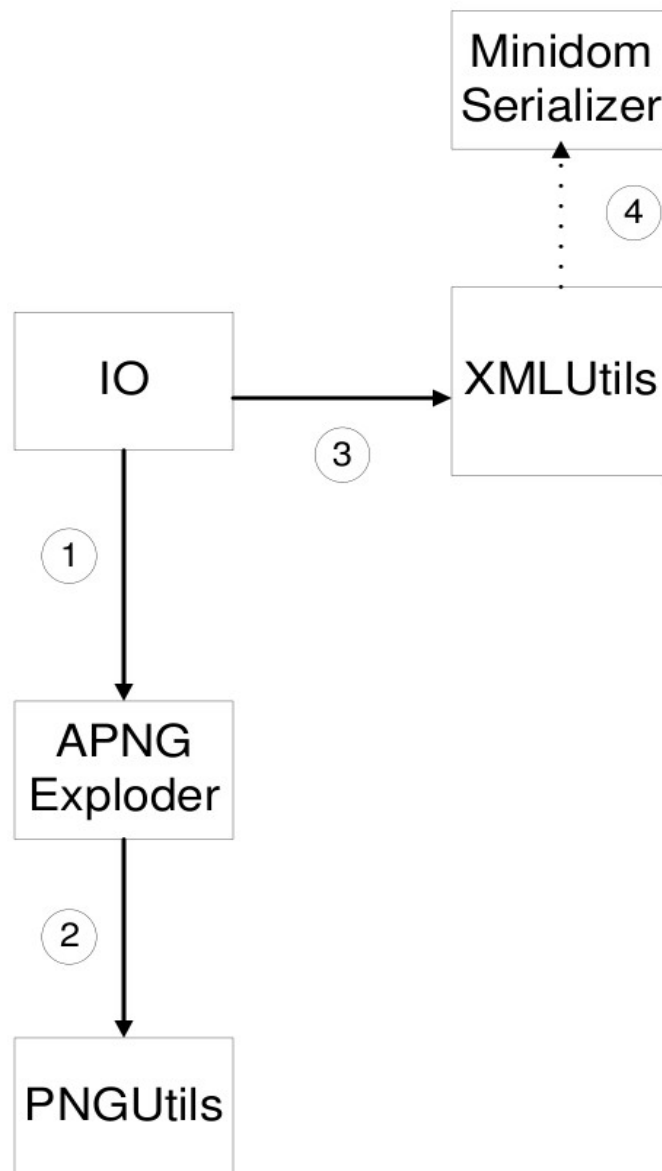
Returns all complete PNG files as a list of byte-lists given the following:

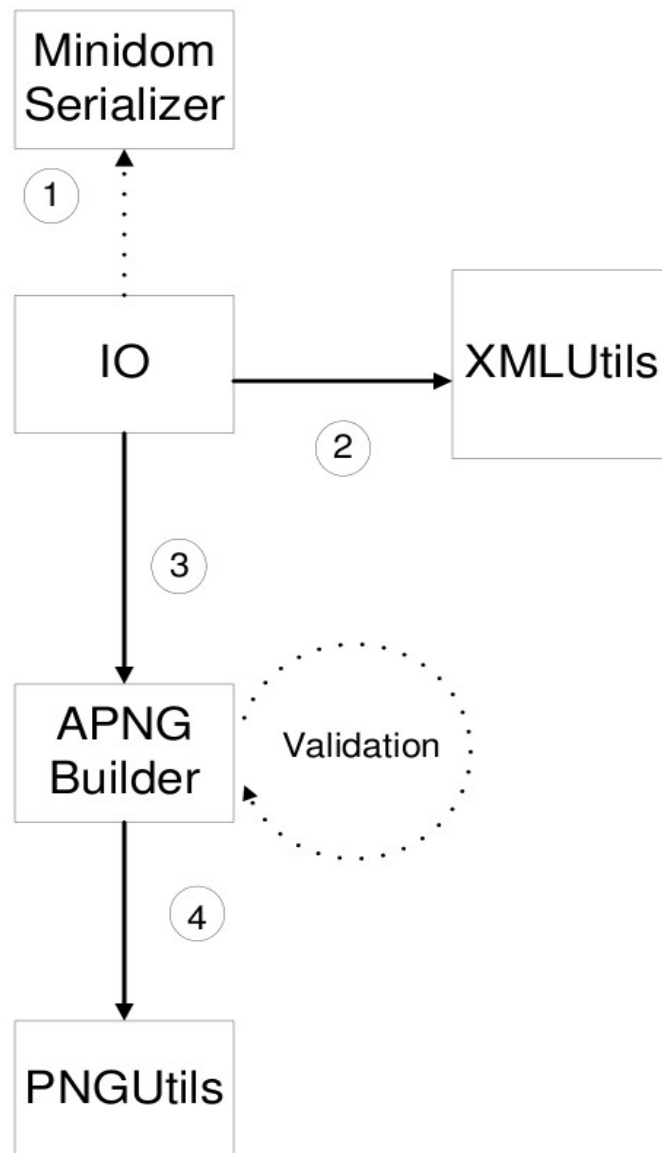
chunks = PNG chunks which represent all PNG files from the input APNG

IDATflag = Defines whether or not the IDAT chunks have been pulled yet

prefix = chunks besides IHDR used for entire APNG file, data contained herein will be used when reconstructing every PNG file.

ihdr = the IHDR chunk data as a byte-list

APNG → PNG's

PNG's → APNG

PROBE Estimate

Avg. LOC	<i>Tiny (7%)</i> 2	<i>Small(24%)</i> 4	<i>Medium(38%)</i> 7	<i>Large(24%)</i> 18	<i>Huge(7%)</i> 43
-----------------	-----------------------	------------------------	-------------------------	-------------------------	-----------------------

Base LOC:

MindomParser 317

MinidomSerializer 125

BasicLex 154

New LOC:

animate(Huge) 43 suspend(Huge) 43

writeFiles(Large) 18 parseXML(Large) 18

getFrames(Med) 7 writeXML(Large) 18

writeFrames(Large) 18 buildAPNG(Huge) 43

preparePNGs(Large) 18 validateIHDR(Large) 18

buildFrames(Large) 18 buildACTL(Med) 7

buildFCTL(Large) 18 blowChunks(Huge) 43

makeChunk(Huge) 43 makeNum(Med) 7

parseNum(Med) 7 calcCRC32(Large) 18

updateCRC32(Large) 18 bytep(Tiny) 2

byte-listp(Small) 4 chunktypep(Tiny) 2

chunkp(Small) 4 chunk-listp(Small) 4

crc32Lookup(Tiny)	2	ascii->bytes(Small)	4
bytes->ascii(Small)	4	explodeAPNG(Huge)	43
getIHDR(Med)	7	getAcTL(Medium)	7
getFrames(Medium)	7		

Contracts

IO – 2 Large cons – 36

XMLConfigReader – 4 Large cons - 72

PNGUtils – 9 Huge cons – 387

APNGBuilder – 5 Huge cons – 215

APNGExploder – 3 Huge cons – 129

Optional: Compression Module → 500 LOC

Final Project Statistics

Total Invested Time Recorded

... defect fixing time: 55 hours 52 minutes

... total time: 252 hours 19 minutes

LoC By Module

Module	APNG Builder	APNG Exploder	basicLex	IO	Minidom Parser	Minidom Serializer	pngUtils	xmlUtils	Total
LoC	187	192	153	124	206	239	260	85	1,446

***Appendix:**

PNG and APNG Specifications

PNG Specifications

Structure of a very simple PNG file

89 50 4E 47 0D 0A 1A 0A PNG signature	IHDR Image header	IDAT Image data	IEND Image end
--	----------------------	--------------------	-------------------

Source: <http://en.wikipedia.org/wiki/APNG>

IHDR ([source](#)):

IHDR Chunk defined as follows:

Width	4 bytes
Height	4 bytes
Bit depth	1 byte
Colour type	1 byte
Compression method	1 byte
Filter method	1 byte
Interlace method	1 byte

-Bit Depth and Color Type

Table 11.1 — Allowed combinations of colour type and bit depth

PNG image type	Colour type	Allowed bit depths	Interpretation
Greyscale	0	1, 2, 4, 8, 16	Each pixel is a greyscale sample
Truecolour	2	8, 16	Each pixel is an R,G,B triple
Indexed-colour	3	1, 2, 4, 8	Each pixel is a palette index; a PLTE chunk shall appear.
Greyscale with alpha	4	8, 16	Each pixel is a greyscale sample followed by an alpha sample.
Truecolour with alpha	6	8, 16	Each pixel is an R,G,B triple followed by an alpha sample.

-Compression Method (zlib)

The only compression method currently used in PNG is 0, or zlib deflate/inflate compression with a sliding window. This compression uses a combination of LZ77 compression with Huffman Coding.

zlib data format is as follows:

zlib compression method/flags code	1 byte
Additional flags/check bits	1 byte
Compressed data blocks	n bytes

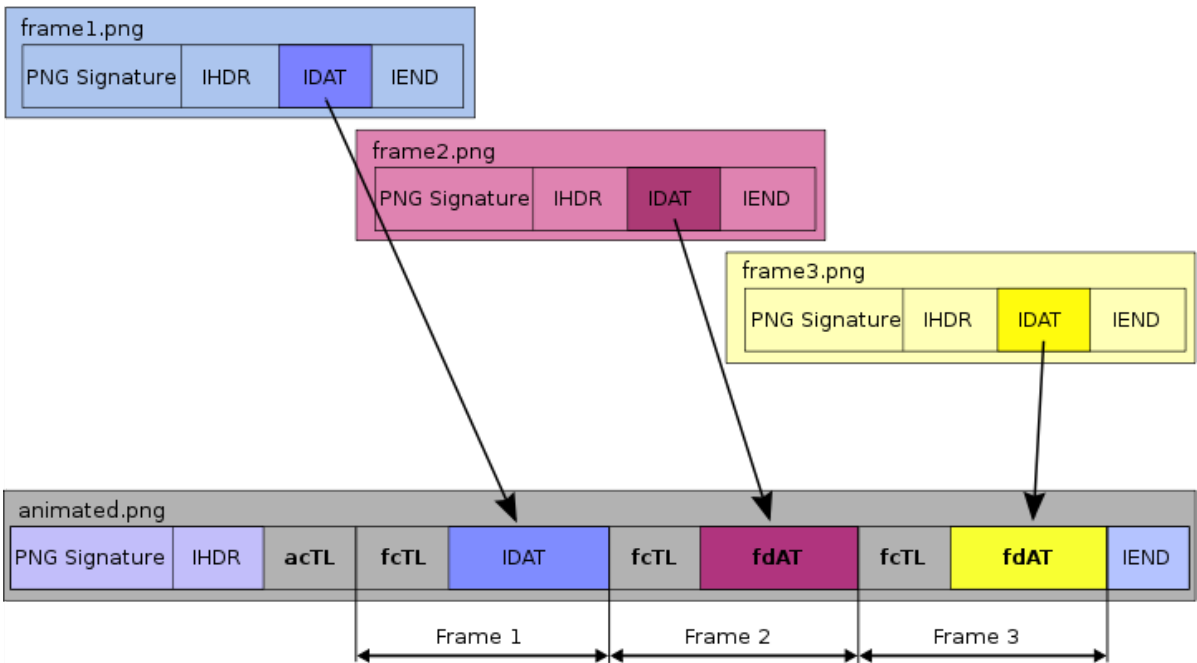
Check value4 bytes

-Filter Method

- x the byte being filtered;
- a the byte corresponding to x in the pixel immediately before the pixel containing x (or the byte immediately before x, when the bit depth is less than 8);
- b the byte corresponding to x in the previous scanline;
- c the byte corresponding to b in the pixel immediately before the pixel containing b (or the byte immediately before b, when the bit depth is less than 8).

Table 9.1 — Filter types			
Type	Name	Filter Function	Reconstruction Function
0	None	$Filt(x) = Orig(x)$	$Recon(x) = Filt(x)$
1	Sub	$Filt(x) = Orig(x) - Orig(a)$	$Recon(x) = Filt(x) + Recon(a)$
2	Up	$Filt(x) = Orig(x) - Orig(b)$	$Recon(x) = Filt(x) + Recon(b)$
3	Average	$Filt(x) = Orig(x) - floor((Orig(a) + Orig(b)) / 2)$	$Recon(x) = Filt(x) + floor((Recon(a) + Recon(b)) / 2)$
4	Paeth	$Filt(x) = Orig(x) - PaethPredictor(Orig(a), Orig(b), Orig(c))$	$Recon(x) = Filt(x) + PaethPredictor(Recon(a), Recon(b), Recon(c))$

APNG Creation Diagram



Source: http://en.wikipedia.org/wiki/APNG#Technical_details