Animator Pro File Formats

This chapter details each of the file formats defined by Animator Pro. Formats supported by Animator Pro but defined by another party, such as GIF and TIFF files, are not described in this document.

The files created by Autodesk Animator Pro are a superset of those created by the original Autodesk Animator. In some cases the Animator Pro formats are identical with the older Animator format. In other cases, new data fields or data compression techniques have been added to the file.

All two-byte and four-byte data values in Animator Pro files are stored in Intel-style order, the same as they would appear in memory on an 80x86 machine.

Common Hierarchial Chunked File Structure

In general, Animator Pro files contain one or more chunks of information. Conceptually, a chunk is a combination of control information and data. The most common Animator Pro file format is a header structure followed by hierarchial data chunks.

Every chunk begins with a header of at least 6 bytes. The first four bytes contain the length of the chunk (including the header itself, and also including the length of all subordinate chunks, if any). The next two bytes are an identifier word which describes the type of data in the chunk. Some chunks have headers longer than six bytes, but the size and identifier fields always occupy the first six bytes of the header.

The Animator Pro animation file is a good example of a hierarchial chunked file structure. The data in an animation file is arranged as follows:

animation file:

- optional prefix chunk:
    - settings chunk
    - cel placement chunk
- frame 1 chunk:
    - postage stamp chunk: postage stamp data
    - color palette chunk
    - pixel data chunk
- frame 2 chunk:
    - pixel data chunk
- frame 3 chunk:
    - color palette chunk
    - pixel data chunk

- frame 4 chunk:
    - color palette chunk
- ring frame chunk:
    - color palette chunk
    - pixel data chunk

It is possible that new types of chunks not described in this document will be added to animation files in the future. We recommend that you quietly ignore unknown chunk types you encounter during animation playback. The size fields in the chunk headers make it easy to skip an entire unrecognized chunk.

FLC - Animator Pro Flic Files

This is the main animation file format created by Animator Pro. The file contains a 128-byte header, followed by an optional prefix chunk, followed by one or more frame chunks.

The prefix chunk, if present, contains Animator Pro settings information, CEL placement information, and other auxiliary data.

A frame chunk exists for each frame in the animation. In addition, a ring frame follows all the animation frames. Each frame chunk contains color palette information and/or pixel data.

The ring frame contains delta-compressed information to loop from the last frame of the flic back to the first. It can be helpful to think of the ring frame as a copy of the first frame, compressed in a different way. All flic files will contain a ring frame, including a single-frame flic.

The FLC file header

A FLC file begins with a 128-byte header, described below. All lengths and offsets are in bytes. All values stored in the header fields are unsigned.

Offset	Length	Name	Description
0	4	size	The size of the entire animation file, including this file header.
4	2	magic	File format identifier. Always hex AF12.
6	2	frames	Number of frames in the flic. This count does not include the ring frame. FLC files have a maximum length of 4000 frames.
8	2	width	Screen width in pixels.
10	2	height	Screen height in pixels.
12	2	depth	Bits per pixel (always 8).
14	2	flags	Set to hex 0003 after ring frame is written and flic header is updated. This indicates that the file was properly

			finished and closed.
16	4	speed	Number of milliseconds to delay between each frame during playback.
20	2	reserved	Unused word, set to 0.
22	4	created	The MSDOS-formatted date and time of the file's creation.
26	4	creator	The serial number of the Animator Pro program used to create the file. If the file was created by some other program using the FlicLib development kit, this value is hex 464C4942 ("FLIB").
30	4	updated	The MSDOS-formatted date and time of the file's most recent update.
34	4	updater	Indicates who last updated the file. See the description of creator.
38	2	aspectx	The x-axis aspect ratio at which the file was created.
40	2	aspecty	The y-axis aspect ratio at which the file was created. Most often, the x:y aspect ratio will be 1:1. A 320x200 flic has a ratio of 6:5.
42	38	reserved	Unused space, set to zeroes.
80	4	oframe1	Offset from the beginning of the file to the first animation frame chunk.
84	4	oframe2	Offset from the beginning of the file to the second animation frame chunk. This value is used when looping from the ring frame back to the second frame during playback.
88	40	reserved	Unused space, set to zeroes.

The FLC prefix chunk

An optional prefix chunk may immediately follow the animation file header. This chunk is used to store auxiliary data which is not directly involved in the animation playback. The prefix chunk starts with a 16-byte header (identical in structure to a frame header), as follows:

Offset	Length	Name	Description
0	4	size	The size of the prefix chunk, including this header and all subordinate chunks that follow.
4	2	type	Prefix chunk identifier. Always hex F100.
6	2	chunks	Number of subordinate chunks in the prefix chunk.
8	8	reserved	Unused space, set to zeroes.

To determine whether a prefix chunk is present, read the 16-byte header following the file

header. If the type value is hex F100, it's a prefix chunk. If the value is hex F1FA it's the first frame chunk, and no prefix chunk exists. Note

Programs other than Animator Pro should never need to create flic files that contain a prefix chunk. Programs reading a flic file should skip the prefix chunk by using the size value in the prefix header to read and discard the prefix, or by seeking directly to the first frame using the oframe1 field from the file header.

The FLC frame chunks

Frame chunks contain the pixel and color data for the animation. A frame chunk may contain multiple subordinate chunks, each containing a different type of data for the current frame. Each frame chunk starts with a 16-byte header that describes the contents of the frame:

Offset	Length	Name	Description
0	4	size	The size of the frame chunk, including this header and all subordinate chunks that follow.
4	2	type	Frame chunk identifier. Always hex F1FA.
6	2	chunks	Number of subordinate chunks in the frame chunk.
8	8	reserved	Unused space, set to zeroes.

Immediately following the frame header are the frame's subordinate data chunks. When the chunks count in the frame header is zero, it indicates that this frame is identical to the previous frame. This implies that no change is made to the screen or color palette, but the appropriate delay is still inserted during playback.

Each data chunk within a frame chunk is formatted as follows:

Offset	Length	Name	Description
0	4	size	The size of the chunk, including this header.
4	2	type	Data type identifier.
6	(size-6)	data	The color or pixel data.

The type values in the chunk headers indicate what type of graphics data the chunk contains and which compression method was used to encode the data. The following values (and their associated mnemonic names) are currently found in frame data chunks:

Value	Name	Description
4	FLI_COLOR256	256-level color palette info
7	FLI_SS2	Word-oriented delta compression
11	FLI_COLOR	64-level color palette info
12	FLI_LC	Byte-oriented delta compression
13	FLI_BLACK	Entire frame is color index 0
15	FLI_BRUN	Byte run length compression
16	FLI_COPY	No compression
18	FLI_PSTAMP	Postage stamp sized image

The following sections describe each of these data encoding methods in detail.

Chunk Type 4 (FLI_COLOR256) - 256-Level Color

The data in this chunk is organized in packets. The first word following the chunk header is a count of the number of packets in the chunk. Each packet consists of a one-byte color index skip count, a one-byte color count and three bytes of color information for each color defined.

At the start of the chunk, the color index is assumed to be zero. Before processing any colors in a packet, the color index skip count is added to the current color index. The number of colors defined in the packet is retrieved. A zero in this byte indicates 256 colors follow. The three bytes for each color define the red, green, and blue components of the color in that order. Each component can range from 0 (off) to 255 (full on). The data to change colors 2,7,8, and 9 would appear as follows:

```
2                       ; two packets
2,1,r,g,b               ; skip 2, change 1
4,3,r,g,b,r,g,b,r,g,b   ; skip 4, change 3
```

Chunk Type 11 (FLI_COLOR) - 64-Level Color

This chunk is identical to FLI_COLOR256 except that the values for the red, green and blue components are in the range of 0-63 instead of 0-255.

Chunk Type 13 (FLI_BLACK) - No Data

This chunk has no data following the header. All pixels in the frame are set to color index 0.

Chunk Type 16 (FLI_COPY) - No Compression

This chunk contains an uncompressed image of the frame. The number of pixels following the chunk header is exactly the width of the animation times the height of the animation. The data starts in the upper left corner with pixels copied from left to right and then top to bottom. This type of chunk is created when the preferred compression method (SS2 or BRUN) generates more data than the uncompressed frame image; a relatively rare situation.

Chunk Type 15 (FLI_BRUN) - Byte Run Length Compression

This chunk contains the entire image in a compressed format. Usually this chunk is used in the first frame of an animation, or within a postage stamp image chunk.

The data is organized in lines. Each line contains packets of compressed pixels. The first line is at the top of the animation, followed by subsequent lines moving downward. The number of lines in this chunk is given by the height of the animation.

The first byte of each line is a count of packets in the line. This value is ignored, it is a holdover from the original Animator. It is possible to generate more than 255 packets on a line. The width of the animation is now used to drive the decoding of packets on a line; continue reading and processing packets until width pixels have been processed, then proceed to the next line.

Each packet consist of a type/size byte, followed by one or more pixels. If the packet type is negative it is a count of pixels to be copied from the packet to the animation image. If the packet type is positive it contains a single pixel which is to be replicated; the absolute value of the packet type is the number of times the pixel is to be replicated.

Chunk Type 12 (FLI_LC) - Byte Aligned Delta Compression

This chunk contains the differences between the previous frame and this frame. This compression method was used by the original Animator, but is not created by Animator Pro. This type of chunk can appear in an Animator Pro file, however, if the file was originally created by Animator, then some (but not all) frames were modified using Animator Pro.

The first 16-bit word following the chunk header contains the position of the first line in the chunk. This is a count of lines (down from the top of the image) which are unchanged from the prior frame. The second 16-bit word contains the number of lines in the chunk. The data for the lines follows these two words.

Each line begins with two bytes. The first byte contains the starting x position of the data on the line, and the second byte the number of packets for the line. Unlike BRUN compression, the packet count is significant (because this compression method is only used on 320x200 flics).

Each packet consists of a single byte column skip, followed by a packet type/size byte. If the packet type is positive it is a count of pixels to be copied from the packet to the animation image. If the packet type is negative it contains a single pixel which is to be replicated; the absolute value of the packet type gives the number of times the pixel is to be replicated.

Note

The negative/positive meaning of the packet type bytes in LC compression is reversed from that used in BRUN compression. This gives better performance during playback.

Chunk Type 7 (FLI_SS2) - Word Aligned Delta Compression

This format contains the differences between consecutive frames. This is the format most often used by Animator Pro for frames other than the first frame of an animation. It is similar to the line coded delta (LC) compression, but is word oriented instead of byte oriented. The data is organized into lines and each line is organized into packets.

The first word in the data following the chunk header contains the number of lines in the chunk. Each line can begin with some optional words that are used to skip lines and set the last byte in the line for animations with odd widths. These optional words are followed by a count of the packets in the line. The line count does not include skipped lines.

The high order two bits of the word is used to determine the contents of the word.

Bit 15	Bit 14	Meaning
0	0	The word contains the packet count. The packets follow this word. The packet count can be zero; this occurs when only the last pixel on a line changes.
1	0	The low order byte is to be stored in the last byte of the current line. The packet count always follows this word.
1	1	The word contains a line skip count. The number of lines skipped is given by the absolute value of the word. This word can be followed by more skip counts, by a last byte word, or by the packet count.

The packets in each line are similar to the packets for the line coded chunk. The first byte of each packet is a column skip count. The second byte is a packet type. If the packet type is positive, the packet type is a count of words to be copied from the packet to the animation image. If the packet type is negative, the packet contains one more word which is to be replicated. The absolute value of the packet type gives the number of times the word is to be replicated. The high and low order byte in the replicated word do not necessarily have the same value.

Chunk Type 18 (FLI_PSTAMP) - Postage Stamp Image

This chunk type holds a postage stamp -- a reduced-size image -- of the frame. It generally appears only in the first frame chunk within a flic file.

When creating a postage stamp, Animator Pro considers the ideal size to be 100x63 pixels. The actual size will vary as needed to maintain the same aspect ratio as the original.

The pixels in a postage stamp image are mapped into a six-cube color space, regardless of the color palette settings for the full frame image. A six-cube color space is formed as follows:

```
start at palette entry 0
for red = 0 thru 5
     for green = 0 thru 5
          for blue = 0 thru 5
               palette_red   = (red   * 256)/6
               palette_green = (green * 256)/6
               palette_blue  = (blue  * 256)/6
               move to next palette entry
          end for blue
     end for green
end for red
```

Any arbitrary rgb value (where each component is in the range of 0-255) can be mapped into the six-cube space using the formula:

((6*red)/256)*36 + ((6*green)/256)*6 + ((6*blue)/256)

When a frame data chunk has been identified as a postage stamp, the header for the chunk contains more fields than just size and type. The full postage stamp chunk header is defined as follows:

Offset	Length	Name	Description
0	4	size	The size of the postage stamp chunk, including this header.
4	2	type	Postage stamp identifier; always 18.
6	2	height	Height of the postage stamp image, in pixels.
8	2	width	Width of the postage stamp image, in pixels.
10	2	xlate	Color translation type; always 1, indicating six-cube color space.

Immediately following this header is the postage stamp data. The data is formatted as a chunk with standard size/type header. The type will be one of:

Value	Name	Description
15	FPS_BRUN	Byte run length compression
16	FPS_COPY	No compression
18	FPS_XLAT256	Six-cube color xlate table

The FPS_BRUN and FPS_COPY types are identical to the FLI_BRUN and FLI_COPY encoding methods described above.

The FPS_XLAT256 type indicates that the chunk contains a 256-byte color translation table instead of pixel data. To process this type of postage stamp, read the pixel data for the full-sized frame image, and translate its pixels into six-cube space using a lookup in the 256-byte color translation table. This type of postage stamp appears when the size of the animation frames is smaller than the standard 100x63 postage stamp size.

FLI - Original Animator Flic Files

This animation file format is limited to 320x200 resolution. It is the main animation file format of the original Animator, and is still used by Animator Pro for creating 320x200 animations. The file structure is very similar to that of a FLC file. A FLI file does not contain a prefix chunk, and does not use FLI_PSTAMP or FLI_SS2 data encoding in the frame chunks.

The FLI file header

The file header for a FLI file is a subset of the FLC file header. It is defined as follows:

Offset	Length	Name	Description
0	4	size	The size of the entire animation file, including this file header.
4	2	magic	File format identifier. Always hex AF11.
6	2	frames	Number of frames in the flic. This count does not include the ring frame. FLI files have a maximum length of 4000 frames.
8	2	width	Screen width in pixels. This is always 320 in a FLI file.
10	2	height	Screen height in pixels. This is always 200 in a FLI file.
12	2	depth	Bits per pixel (always 8).
14	2	flags	Always zero in a FLI file.
16	2	speed	Number of jiffies to delay between each frame during playback. A jiffy is 1/70 of a second.
18	110	reserved	Unused space, set to zeroes.

The FLI frame chunks

One or more frame chunks immediately follow the FLI file header. The frame chunks in a FLI file are identical to those in a FLC file, except that postage stamp image (FLI_PSTAMP) and word-runlength-compression (FLI_SS2) data chunks never appear in FLI files.

CEL - Animation Cel Files

CEL files contain one or more frames of image data. Both Animator Pro and the original Animator produce CEL files, but each uses a different file format.

To process a CEL file for input, read the first 2 bytes of the file. If they are hex 9119, the file is an original Animator CEL file. If the first two bytes are not 9119, it is an Animator Pro CEL file.

Animator Pro CEL Files

An Animator Pro CEL file is identical to a FLC file in all respects. A CEL file should have a Celdata chunk in the file prefix chunk which describes the x,y placement of the CEL. If the Celdata placement chunk is not present, assume a placement of 0,0.

Original Animator CEL Files

The original Animator also produced CEL files. These were still-picture files, not the multi-frame files Animator Pro now uses. A CEL file from the original Animator is identical to a PIC file from the original Animator in all respects.

PIC - Picture Files

PIC files contain still images in an uncompressed format. Both the original Animator and Animator Pro produce PIC files. The file formats are different; Animator Pro produces a hierarchial chunked file, while the original Animator file is a simpler fixed format. These formats are detailed in the following sections.

To process a PIC file for input, read the first 2 bytes of the file. If they are hex 9119, the file is an original Animator PIC format file. If the first two bytes are not 9119, it is an Animator Pro PIC file.

Animator Pro PIC Files

Animator Pro uses this format to store a single-frame picture image or bitmap. This format description applies to both PIC and MSK files. The file begins with a 64-byte header defined as follows:

Offset	Length	Name	Description
0	4	size	The size of the file, including this header.
4	2	magic	File format identifier. Always hex 9500.
6	2	width	The width of the image, in pixels.
8	2	height	The height of the image, in pixels.
10	2	xcoord	The X coordinate; typically zero. (See note below).
12	2	ycoord	The Y coordinate; typically zero. (See note below).
14	4	userid	An arbitrary 4-byte value; generally zero. Do not count on any particular value in this field. Set this field to

			zero when creating a file.
18	1	depth	The number of bits per pixel. This is 8 for PIC files and 1 for MSK files.
19	45	reserved	Unused space; set to zeroes.

Note

The xcoord and ycoord values in a PIC file header will typically be zero. Non-zero values indicate that the file contains a rectangle from within a larger picture. In this case, the xcoord and ycoord values represent the relation of the saved rectangle to the full image. These values can be safely ignored for most purposes.

Following the file header are the data chunks for the image. Each data chunk within a PIC or MSK file is formatted as follows:

Offset	Length	Name	Description
0	4	size	The size of the chunk, including this header.
4	2	type	Data type identifier.
6	(size-6)	data	The color or pixel data.

The type values in the chunk headers indicate what type of graphics data the chunk contains. The following values (and their associated mnemonic names) are currently found in PIC/MSK data chunks:

Value	Name	Description
0	PIC_CMAP	Color palette info
1	PIC_BYTEPIXELS	Byte-per-pixel image data
2	PIC_BITPIXELS	Bit-per-pixel mask data

In a PIC_CMAP chunk, the first 2-byte word is a version code; currently this is set to zero. Following the version word are all 256 palette entries in rgbrgb... order. Each of the r, g, and b components is a single byte in the range of 0-255. This type of chunk appears in PIC files; there will generally be no color map chunk in a MSK file.

In a PIC_BYTEPIXELS chunk, the image data appears immediately following the 6-byte chunk header. The data is stored as one byte per pixel, in left-to-right, topD to-bottom sequence. This type of chunk appears in PIC files.

In a PIC_BITPIXELS chunk, the bitmap data appears immediately following the 6-byte chunk header. The data is stored as bits packed into bytes such that the leftmost bits appear in the high-order positions of each byte. The bits are stored in left-to-right, top-toD bottom sequence. When the width of the bitmap is not a multiple of 8, there will be unused bits in the lowD order positions of the last byte on each line. The number of bytes per line is ((width+7)/8). This type of chunk appears in MSK files.

Original Animator PIC Files

The original Animator uses this format to store a single-frame picture image. This format description applies to both PIC and CEL files. The file begins with a 32 byte header, as follows:

Offset	Length	Name	Description

0	2	type	File type identifier. Always hex 9119.
2	2	width	Width of image. Always 320 in a PIC file; may be any value in a CEL file.
4	2	height	Height of image. Always 200 in a PIC file; may be any value in a CEL file.
6	2	xcoord	X coordinate for upper left corner of the image. Always zero in a PIC file; may be non-zero in a CEL file.
8	2	ycoord	Y coordinate for upper left corner of the image. Always zero in a PIC file; may be non-zero in a CEL file.
10	1	depth	Number of bits per pixel; always 8.
11	1	compress	Compression flag; always zero.
12	4	datasize	Size of the image data in bytes.
16	16	reserved	Unused space; set to zeroes.

Immediately following the header is the color map. It contains all 256 palette entries in rgbrgb... order. Each of the r, g, and b components is a single byte in the range of 0-63. Following the color palette is the image data, one byte per pixel. The image data is stored in left-to-right, top-to-bottom sequence.

MSK - Mask Data Files

MSK files contain a bitmap image. Both Animator Pro and the original Animator produce MSK files, but the formats are different.

To process a MSK file for input, check the file size. If it is exactly 8000 bytes, the file is an original Animator MSK file. If the file is any other size, it is an Animator Pro MSK file.

Animator Pro MSK Files

An Animator Pro MSK file is identical to an Animator Pro PIC file. It will have a pixel depth of 1.

Original Animator MSK Files

A MSK file created by the original Animator is exactly 8000 bytes long. There is no file header or other control information in the file. It contains the image bit map, 1 bit per pixel, with the leftmost pixels packed into the high order bits of each byte. The size of the image is fixed at 320x200. The image is stored left-to-right, top-to-bottom.

COL - Color Map Files

A COL file stores the rgb values for entries in the color palette. Both Animator Pro and the original Animator produce COL files, but the formats are different.

To process a COL file for input, check the file size. If it is exactly 768 bytes, the file is an original Animator COL file. If the file is any other size, it is an Animator Pro COL file.

Animator Pro COL Files

An Animator Pro COL file stores color palette information. The file begins with an 8-byte header defined as follows:

Offset	Length	Name	Description
0	4	size	The size of the file, including this header.
4	2	magic	File format identifier. Always hex B123.
6	2	version	The version of color storage format. Currently set to zero, indicating 256-level color data in each r,g,b component.

Following the file header are palette entries in rgbrgb... order. Each of the r, g, and b components is a single byte in the range of 0-255. Generally, there will be data for 256 palette entries, but this cannot be assumed. The actual number of palette entries is ((size-8)/3); if this value is not an even multiple of three, the file is corrupted.

Original Animator COL Files

A COL file created by the original Animator is exactly 768 bytes long. There is no file header or other control information in the file. The rgb values for all 256 palette entries is stored in rgbrgb... sequence. Each of the r, g, and b values is in the range of 0-63.

PLY - Polygon Files

A PLY file holds a set of points that describe a polygon. Both Animator Pro and the original Animator create PLY files. The file format is the same for both.

A PLY file starts with an 8-byte header, as follows:

Offset	Length	Name	Description
0	2	points	Count of points in the file.
2	4	reserved	Unused space; set to zero.
6	1	closed	Closed-shape flag. If 1, there is an implied connection between the last point and the first. If zero, the shape is not closed.
7	1	magic	File format identifier. Always hex 99.

The points data follows the file header. Each point is described with three 16-bit integers, representing the x, y, and z coordinates of each point. The z coordinates are always zero.

TWE - Tween Data Files

A TWE file holds information about a tweening operation set up via the Tween menus. The information includes the starting and ending shapes, and the optional userD specified links between the shapes. Animator Pro creates tween files.

A TWE file begins with an 8-byte header defined as follows:

Offset	Length	Name	Description
0	2	magic	File format identifier. Always hex 1995.
2	2	version	The file format version; always zero.
4	4	tcount	The number of tween shapes in the file; always 2.
8	8	reserved	Unused space; set to zeroes.
16	4	linkcount	The number of link entries in the file.

Immediately following the file header are the link entries. If the linkcount value is zero there are no links. Each link entry is a pair of 32-bit integers. The first value in each pair is the index of the point in the first shape, and the second value is the index of the point in the ending shape. (IE, a link value of 2,7 says to link the second starting-shape point to the seventh ending-shape point.)

Following the link entries is the data block that describes the starting shape, then the data block that describes the ending shape. The format of these blocks is identical to that of the polygon (PLY) file, including file header data. In other words, they appear as if a pair of polygon files are embedded in the tween file at this point.

OPT - Optics Menu Settings Files

An OPT file holds information about an optics operation set up via the Optics menus. Both Animator Pro and the original Animator create OPT files. The file format is the same for both.

An OPT file starts with a 4-byte header, as follows:

Offset	Length	Name	Description
0	2	magic	File type identifier. Always hex 1A3F.
2	2	count	Number of records in the file.

Following the file header are optics records of 50 bytes each. A record is generated for each click on CONTINUE MOVE in the OPTICS menu. The move records are formatted as follows:

Offset	Length	Name	Description
0	4	link	In the file, this field is always zero. In memory, it's a pointer to the next move record.
4	6	spincenter	The x,y,z coordinates of the spin center point; three 16-bit values.
10	6	spinaxis	The x,y,z coordinates of the spin axis; three 16-bit values.
16	6	spinturns	The x,y,z coordinates of the spin turns; three 16-bit values.
22	4	spininter	Intermediate turns. Two 16-bit values. These are values for a conjugation matrix that corresponds to spin axis.

26	6	sizecenter	The x,y,z coordinates of the size center point; three 16-bit values.
32	2	xmultiplier	Determines (along with xdivisor) how to scale along x dimension.
34	2	xdivisor	Determines (along with xmultiplier) how to scale along x dimension.
36	2	ymultiplier	Determines (along with ydivisor) how to scale along y dimension.
38	2	ydivisor	Determines (along with ymultiplier) how to scale along y dimension.
40	2	bothmult	Like xmultiplier, but applied to both dimensions.
42	2	bothdiv	Like xdivisor, but applied to both dimensions.
44	6	linearmove	The x,y,z offset for a linear move; three 16-bit values.

Internal Usage Files (REC, SET, CFG, GLV, MU)

Each of these file types is created by Animator Pro to store internal data between sessions. These files must not be touched by other applications.

REC files store macros. Animator Pro REC files are not the same as the REC files documented with the original Animator.

SET files store internal settings information; they are created by the Quit-Save menu, and by the Save Default Settings menu.

CFG files store internal configuration information, such as the video driver and mode.

GLV files store Poco Global Variables. The data is stored as a series of nullterminated strings, and must not be modified using a normal text editor.

MU files store menu text, prompts, error messages, etc, customized to the proper native language. The data is stored as normal ASCII text, and must not be modified in any way.

End of document.

Below are a few comments that hold text we may want to suck back into the document some day. When the prefix chunk is present, it contains a fixed header structure, followed by one or more prefix sub-chunks. The prefix chunk header is defined as follows:

The following constants identify the sub-chunks within a prefix chunk.

```
enum {
     FP_FREE        = 0,
     FP_FLIPATH          = 1,
     FP_VSETTINGS   = 2,
     FP_CELDATA          = 3,
     };
```

Format for the optional Prefix chunk:

The prefix chunk at present contains:

A settings chunk. This is the same thing as a settings file but as a chunk in a flic. This is how the load settings from a flic is implemented. This is present in flics but not cels. A settings chunk has sub chunks for things like the paths in the file requestors, optics settings, etc.

A Celdata chunk. This chunk has the positioning info for a fli used as a cel. This is usually only present in cel files.