FCEUX Movie File format

FCEUX uses a new movie file format - .fm2.

This differs from the previous FCE Ultra movie format (.fcm) in the following ways:

- It is text based by default; allowing easy movie editing/splicing
- An imbedded GUID so FCEUX can tell if a savestate belongs to a movie file
- Movies recorded from Start (Power-on) no longer have a redundant savestate
- Contains mouse input for recording the Zapper & Arkanoid Paddle

Format

FM2 consists of two parts: Header and Input Log.
The header is always in ASCII plain text format. It consists of several key-value pairs.
The input log section can be identified by it starting with a | (pipe).
The input log section can be either in ASCII plain text format or in binary format.
The input log section terminates at EOF, unless the **length** key is specified in header.
Newlines may be \r\n or \n.

Header

Key-value pairs consist of a key identifier, followed by a space separator, followed by the value text.
Value text is always terminated by a newline, which the value text does

not include.
The value text is parsed differently depending on the type of the key.
The key-value pairs may be in any order, except that the first key must be version.

Integer keys (also used for booleans, with a 1 for true and 0 for false) must have a value that can be stored as int32:

- version (required) - the version of the movie file format; for now it is always 3

- emuVersion (required) - the version of the emulator used to produce the movie

- rerecordCount (optional) - the rerecord count

- **palFlag** (bool) (optional) - true if the movie uses PAL timing

- **NewPPU** (bool) (optional) - true if the movie uses New PPU

- FDS (bool) (optional) - true if movie was recorded on a Famicom Disk System (FDS) game

- fourscore (bool) - true if a fourscore was used. If fourscore is not used, then port0 and port1 are required

- port0 - indicates the type of input device attached to the port 0. Supported values are:
SI_NONE = 0
SI_GAMEPAD = 1
SI_ZAPPER = 2

- port1 - indicates the type of input device attached to the port 1. Supported values are:
SI_NONE = 0
SI_GAMEPAD = 1
SI_ZAPPER = 2

- port2 (required) - indicates the type of the FCExp port device which was attached. Supported values are:
SIFC_NONE = 0

- binary (bool) (optional) - true if input log is stored in binary format

- length (optional) - movie size (number of frames in the input log). If this key is specified and the number is >= 0, the input log ends after specified number of records, and any remaining data should not be parsed. This key is used in **fm3** format to allow storing extra data after the end of input log

String keys have values that consist of the remainder of the key-value pair line. As a consequence, string values cannot contain newlines.

- romFilename (required) - the name of the file used to record the movie

- comment (optional) - simply a memo

- by convention, the first token in the comment value is the subject of the comment
- by convention, subsequent comments with the same subject should have their ordering preserved and may be used to approximate multi-line comments
- by convention, the author of the movie should be stored in comment(s) with a subject of: **author**

Example:

- comment author adelikat

- subtitle (optional) - a message that will be displayed on screen when movie is played back (unless Subtitles are turned off, see Movie options)

- by convention, subtitles begin with the word "subtitle"
- by convention, an integer value following the word "subtitle" indicates the frame that the subtitle will be displayed
- by convention, any remaining text after the integer is considered to be the string displayed

Example:

- subtitle 1000 Level Two

At frame 1000 the words "Level Two" will be displayed on the screen

- guid (required) - a unique identifier for a movie, generated when the movie is created, which is used when loading a savestate to make sure it belongs to the current movie
GUID keys have a value which is in the standard guide format:

452DE2C3-EF43-2FA9-77AC-0677FC51543B

- romChecksum (required) - the base64 of the hexified MD5 hash of the ROM which was used to record the movie

- savestate (optional) - a fcs savestate blob, in case a movie was recorded from savestate
Hex string keys (used for binary blobs) have a value that is like 0x0123456789ABCDEF...

Input log

The input log section consists of movie records either in the form of text lines or in the form of binary data.

Text format (default format):

Every frame of the movie is represented by line of text beginning and ending with a | (pipe).
The fields in the line are as follows, except when fourscore is used.
|commands|port0|port1|port2|

Field **commands** is a variable length decimal integer which is interpreted as a bit field corresponding to miscellaneous input states which are valid at the start of the frame. Current values for this are:

- 1 = Soft Reset
- 2 = Hard Reset (Power)
- 4 = FDS Disk Insert
- 8 = FDS Disk Select
- 16 = VS Insert Coin

The format of port0, port1, port2 depends on which types of devices were attached.

SI_GAMEPAD:

- the field consists of eight characters which constitute a bit field

- any character other than ' ' or '.' means that the button was pressed
- by convention, the following mnemonics are used in a column to remind us of which button corresponds to which column: **RLDUTSBA** (Right, Left, Down, Up, sTart, Select, B, A)

SI_ZAPPER:

- **XXX YYY B Q Z**

XXX: %03d, the x position of the mouse
YYY: %03d, the y position of the mouse
B: %1d, 1 if the mouse button is pressed; 0 if not
Q: %1d, an internal value used by the emulator's zapper code
Z: %d, a variable-length decimal integer; an internal value used by the emulator's zapper code

SI_NONE:

- the field must be empty

If a **fourscore** is used, then port0 and port1 are irrelevant and ignored. The input types must all be gamepads, and each input log record must be in the following format:
|commands|RLDUTSBA|RLDUTSBA|RLDUTSBA|RLDUTSBA
{commands, player1, player2, player3, player4, port2}

Binary format:

Input log section starts with a | (pipe).
Every frame of the movie is represented by a record of a fixed length which can be determined by the devices on port0 and port1.

The first byte of each record stores "commands" bit field.

- bit 0 = Soft Reset
- bit 1 = Hard Reset (Power)
- bit 2 = FDS Disk Insert
- bit 3 = FDS Disk Select
- bit 4 = VS Insert Coin

The remaining bytes in the record depend on which types of devices are attached to port0 and port1.

SI_GAMEPAD:

- 1 byte added to the size of record
- bits of the byte represent the state of buttons (bit0 = A, bit1 = B, bit2 = Select, bit3 = sTart, bit4 = Up, bit5 = Down, bit6 = Left, bit7 = Right). If the bit is set, respective button is considered to be pressed, if the bit is clear, the button is not pressed

SI_ZAPPER:

- 12 bytes added to the size of record
- 1st byte - the x position of the mouse
- 2nd byte - the y position of the mouse
- 3rd byte - 1 if the mouse button is pressed; 0 if not
- 4th byte - an internal value used by the emulator's zapper code
- bytes 5-12 (uint64) - an internal value used by the emulator's zapper code

SI_NONE:

- 0 bytes added to the size of record

If a **fourscore** is used, then port0 and port1 are irrelevant and ignored. 4 bytes are added to the size of record. The bits of the 1st byte represent the state of buttons of the 1st joypad (bit0 = A, bit1 = B, bit2 = Select, bit3 = sTart, bit4 = Up, bit5 = Down, bit6 = Left, bit7 = Right); bits of the 2nd byte represent the state of buttons of the 2nd joypad, and so on.

Notes:

A. All movies start from power-on, unless a savestate key-value is present.

B. The emulator uses these framerate constants
 - NTSC: 1008307711 /256/65536 = 60.099822938442230224609375

- PAL : 838977920 /256/65536 = 50.00698089599609375

FCE Ultra Movie File Format

- Updated March 22, 2004

The FCM file format is a somewhat "joined" file format. The first part of a FCM
file will contain an FCS-format state save. After this data, the FCM-specific data
begins, which is being referred to from this point.

Currently, the only supported input scheme for a FCM is four joysticks.

The FCM data consists of a stream of joystick commands:

```
	dLLjjbbb

	d  = Dummy update, if set.  Used to reset frame timestamp.
	LL  = timestamp length, in bytes(maximum of 3 bytes).
	jj  = Which joystick(0-3).
	bbb = Which button(0-7).

	If the dummy update bit is set, a command can also have occurred.  Look at the
	lower 5 bits:
		0	=	Just a dummy update.
		1	=	Reset
		2	=	Power

	The timestamp is stored after the joystick command, in LSB-first format.  It is
	the number of frames since the last event.  A timestamp length of "0" is valid, to
	be used when several different buttons need to change state at the same time(same frame,
```

at least).

Savestate (.fcs)

FCE Ultra Save State Format
Updated: Mar 9, 2003

FCE Ultra's save state format is now designed to be as forward and backwards
compatible as possible. This is achieved through the (over)use of chunks.
All multiple-byte variables are stored LSB(least significant byte)-first.
Data types:

    (u)int8 - (un)signed 8 bit variable(also referred to as "byte")
    (u)int16 - (un)signed 16 bit variable
    (u)int32 - (un)signed 32 bit variable

-- Main File Header:

The main file header is 16-bytes in length. The first three bytes contain the string "FCS". The next byte contains the version of FCE Ultra that saved
this save state. This document only applies to version "53"(.53) and higher.
After the version byte, the size of the entire file in bytes(minus the 16 byte main file header) is stored. If oldversion is set to 255, the 32-bit version field will be used. In this field, a version such as 0.98.10 is stored as "9810"(decimal).
The rest of the header is currently unused and should be nulled out.
Example of relevant parts:

    FCS <uint8 oldversion> <uint32 totalsize> <uint32 version>

-- Section Chunks:

Sections chunk headers are 5-bytes in length. The first byte defines what

section it is, the next four bytes define the total size of the section
(including the section chunk header).

```
<uint8 section> <uint32 size>
```

Section definitions:

1	-	"CPU"
2	-	"CPUC"
3	-	"PPU"
4	-	"CTLR"
5	-	"SND"
16	-	"EXTRA"

-- Subsection Chunks

Subsection chunks are stored within section chunks. They contain the actual
state data. Each subsection chunk is composed of an 8-byte header and the data.
The header contains a description(a name) and the size of the data contained
in the chunk:

```
<uint8 description[4]> <uint32 size>
```

The name is a four-byte string. It does not need to be null-terminated.
If the string is less than four bytes in length, the remaining unused bytes
must be null.

-- Subsection Chunk Description Definitions

Note that not all subsection chunk description definitions listed below
are guaranteed to be in the section chunk. It's just a list of what CAN
be in a section chunk. This especially applies to the "EXTRA"
subsection.

---- Section "CPU"

Name:	Type:	Description:
PC	uint16	Program Counter
A	uint8	Accumulator
P	uint8	Processor status register
X	uint8	X register
Y	uint8	Y register
S	uint8	Stack pointer
RAM	uint8[0x800]	2KB work RAM

---- Section "CPUC" (emulator specific)

Name:	Type:	Description:
JAMM	uint8	Non-zero value if CPU in a "jammed" state
IRQL	uint8	Non-zero value if IRQs are to be generated constantly
ICoa	int32	Temporary cycle counter
ICou	int32	Cycle counter

---- Section "PPU"

Name:	Type:	Description:
NTAR	uint8[0x800]	2 KB of name/attribute table RAM
PRAM	uint8[32]	32 bytes of palette index RAM
SPRA	uint8[0x100]	256 bytes of sprite RAM
PPU	uint8[4]	Last values written to $2000 and $2001, the PPU status register, and the last value written to $2003.
XOFF	uint8	Tile X-offset.
VTOG	uint8	Toggle used by $2005 and $2006.
RADD	uint16	PPU Address Register(address written to/read from when $2007 is accessed).

TADD	uint16	PPU Address Register
VBUF	uint8	VRAM Read Buffer
PGEN	uint8	PPU "general" latch. See Ki's document.

---- Section "CTLR" (somewhat emulator specific)

Name:	Type:	Description:
J1RB	uint8	Bit to be returned when first joystick is read.
J2RB	uint8	Bit to be returned when second joystick is read.

---- Section "SND" (somewhat emulator specific)

NREG	uint16	Noise LFSR.
P17	uint8	Last byte written to $4017.
PBIN	uint8	DMC bit index.
PAIN	uint32	DMC address index(from $8000).
PSIN	uint32	DMC length counter(how many bytes left to fetch).

<to be finished>

---- Section "EXTRA" (varying emulator specificness)

For iNES-format games(incomplete, and doesn't apply to every game):

Name:	Type:	Description:
WRAM	uint8[0x2000]	8KB of WRAM at $6000-$7fff
MEXR	uint8[0x8000]	(very emulator specific)
CHRR	uint8[0x2000]	8KB of CHR RAM at $0000-$1fff(in PPU address space).
EXNR	uint8[0x800]	Extra 2KB of name/attribute table RAM.
MPBY	uint8[32]	(very emulator specific)

MIRR	uint8	Current mirroring: 0 = "Horizontal" 1 = "Vertical" $10 = Mirror from $2000 $11 = Mirror from $2400
IRQC	uint32	Generic IRQ counter
IQL1	uint32	Generic IRQ latch
IQL2	uint32	Generic IRQ latch
IRQA	uint8	Generic IRQ on/off register.
PBL	uint8[4]	List of 4 8KB ROM banks paged in at $8000-$FFFF
CBL	uint8[8]	List of 8 1KB VROM banks page in at $0000-$1FFF(PPU).

For FDS games(incomplete):

Name:	Type:	Description:
DDT<x>	uint8[65500]	Disk data for side x(0-3).
FDSR	uint8[0x8000]	32 KB of work RAM
CHRR	uint8[0x2000]	8 KB of CHR RAM
IRQC	uint32	IRQ counter
IQL1	uint32	IRQ latch
IRQA	uint8	IRQ on/off.
WAVE	uint8[64]	Carrier waveform data.
MWAV	uint8[32]	Modulator waveform data.
AMPL	uint8[2]	Amplitude data.