ENSONIQ EPS SEQUENCE STRUCTURE

This document describe the organization of files for sequences (and songs for the eps classic and the eps 16+ samplers. These two generation of samplers use different file types. The 16+ can convert a classic sequence to its own format while the classic cannot read songs or sequences from the 16+. There are many similarities in the structure of the sequences. This document is divided into sections, the headers of which should indicate whether it applies only to 16+ or to both samplers. It will probably be beneficial to read all sections on the classic even if you are only interested in the 16+. The eps-classic case is presented first, then what is different in the 16 + case is presented. Some familiarity with the EPS will be required.

Sequences can be stored individually on disk, or several sequences can b stored together in a SONG. In these cases the structure of the sequence chunk is almost the same. We shall call these sequence types a STANDALONE SEQUENCE and a LINKED SEQUENCE respectively. In addition, a song has tracks running for the whole length of a song, the structure of these are the same as that for the standalone sequence. We will also call these song tracks for a SONG SEQUENCE. Finally there is a structure in a song which only has some similarities with the other sequences and that is the SONG STEPS.

In order to take direct advantage of the information that is given here yo first have to access files individually (as found in the directory of an eps disk). The needed information for doing so can be found several places(see footnote 2).

A note on 12-bit integers and vectors/longs and number notation

Some numbers found in eps sequences may at first appear strange when viewe by a hex-dump. They may perhaps make a lot more sense if you assume they are made the way they are in order to load efficiently into a Ò12 bit wide memoryÓ on a machine using a processor with 16 bit word size. There are many ways of doing this and Ensoniq have used a few of them for the EPS. Here we describe one common description occurring in Sequences: An integer number that will indicate the hex number $1A will be stored as $01A0. The least significant nybble of the integer is not used. You see that you will obtain the intended value of the integer by dividing its file representation by 16. I sometime notate variables stored this way as integerX16. Sometimes one need numbers larger than 2^12 and a longint occupying 4 bytes in the file will be used. The number is then stored as a pair of integerX16 with the least significant integerX16 first. Example: the address $1234 is stored as $2340 0010. We may call this last type as Òeps_longint.Ó

When I in this document write $1234 I use Ò$Ó to indicate hexadecimal, use %110011 to indicate binary and I use 20. to indicate decimal twenty. The default is hex however. Ordering of integers, bytes, nybbles, and bits refers to the order they have in the file. The numbering starts at zero. The second byte is byte number one. The first byte, bit or nybble occurring when the number is handwritten is referred to as the most significant one.

ÒHeader structureÓ,classic

(in the tables the sizes and offsets (ofs) are in bytes unless otherwis specified,)

Ofs	Size	VarName	Meaning/Comment
0	4	SeqLen	Length of sequence in bytes stored as eps_longin
4	24.	SeqName	Name of seq Only the first byte of each integer is used, the second is always zero except the 20th byte which is $30 for a standalone and linked sequenc (but is 0 for a song sequence

```
                    For a song sequence the msb of all integers are $2E
 1C  2    NumBars    number of bars x16
 1E  2    TimeSig    time signature
                     bit5-8-> nominator, bit 9-11 deno
                      denom: 2=quarter,3=eight,4=sixteenth, 1=half 0=whol
                      example $05C0 = %1011 1.100 0000 =11/1
 20  2     Tempo      beats pr. min. X16, least sig. nybble not used
                      most sig. nybble not used, that is loop-flag
 20  1    LoopFlg     Most sig nybble used, off=$Fx, on=0x, NB shared
 22  4  StartInfoTrk  Offset address of info track, eps_lon
 22 4X9  StartTrkTbl
 26  4   StartTrk1    Start address from top of file track
 2A  4   StartTrk2    -----------Ó ----------------------
 2E  4   StartTrk3    -----------Ó ----------------------
 32  4   StartTrk4    -----------Ó ----------------------
 36  4   StartTrk5    -----------Ó ----------------------
 3A  4   StartTrk6    -----------Ó ----------------------
 3E  4   StartTrk7    -----------Ó ----------------------
 42  4   StartTrk8    -----------Ó ----------------------

 46        here is the normal  start for infotrk = trk
 68        here is the normal start for first recorded trac

 Track_Chunk structure classic and 16 plu
```

In a sequence there appear to be possible to have 9 Track Chunks all havin essentially the same structure. The beginning address of the chunks are found in StartInfoTrk, and StartTrkn where n=1,2,3 ..8. The exact meaning of what is called InfoTrk may not be known, but effectively it only has a advance_time-command containing the total number of clicks in the sequence. The other tracks are simply the contents of the tracks as played back on the EPS by soloing that particular track.

The structure starts with a eps_long (with the least significan integerX16 first) that holds the number of bytes that there is in the sequence. Then follows two integers which are always? zero. Then follows 20 bytes that are not used. It is guessed that it can function as some sort of scratch area. The actual contents of the track_chunk then starts at offset 28. from the top of the track. The content will consist of integers grouped together 1 ,2 ,3 or 4 depending upon its function. A Track ends with the EndOfTrack message ($8BC0 for classic while $80E9 for plus) The end is also given implicit by the length of course.

OK so here in schematic form

Track_chunk classic and 16

```
 ofs size  name      meaning/comment
 0    4   TrkLength  Byte length of Track stored in an eps_lon
 4    4   unknown ,  set it to
 8    20. Scratch    Unknown, Not used when read/writing to flopp
 $1C  ..  eventstart A sequence of messages,
                     ended by the EndOfTrack command.
```

ÒSequencer event structureÓ-classi

It may be appropriate to regard the sequencer events as a collection o messages. The length of a complete message (in the file)is always an even number of bytes-so we can also regard each message to consist of integers. The number of integers contained in a message is dependent upon what the message is about. We can say the first integer of the message always contain information on what kind of command (note, program, mod wheel ..) it is. There may be zero to 5 data integers following the command identifying integer. The length of the message is fixed for a given command. See ÕparsingÕ for how to determine if an integer is a command, and if it is, what type it is. The most significant nybble is irrelevant for

the type of command . Nybble 1 and 2 constitute a byte determining the command.(and nybble 3 is 0) Example: The first integer of the message is $ABC0 -here $BC is a byte identifying the command. $A has the most sig. bit set signalling that it is a command. The three least sig. bits of the nybble $A are time-clocks. For now we can ignore its value This means that both BBC0 and 8BC0 would signal the same command. (just the timing of the next command is different) .See below.

Timing of events, classi

A message contain information on when the next event is to occur (not the timing of itself). This time information is contained in the three least significant bits of the most significant nybble of the command integer. In addition, if the message contain a data-integer then quite frequently (a few exceptions exist) the 5 most significant bits of the last data integer (however the first bit is 0) also contain timing info. Example: message ABD0 1C00 , Here the command-integer is ABD0.. ItÕs most significant nybble is A=%1010, so the time-bits are %010=$2. The last data integer is 1C00=%0001 1100 0000 0000. The 5 most sig. bits here are %00011=$3. So the next event will occur after $23 clock-pulses. It is known the eps uses time-clocks of which there are 48. to the quarter-note. Lets notate the time delay until next event by the symbol dClk(delta clock). Trk0 or the infotrack contains a time command. It contains the exact number of clocks in the sequence. The sum of all dClks that is given in a track should add up to the value given in the infotrack.

The different commands, classi

As noted above the second and third nybble (nybble 1 and 2, $BC in $ABC0 contain the key to identifying the command. First we present a list of these command keys, then we list the different values for these commands, indicate how many bytes a message of that type will contain and give an example of a message and decode it.

list of command

00-57	note messag
58-AF	after-touch, key pressur
B0	pitch wheel, midi pitch bend
B1	modulation wheel, midi ctl 1
B2	patch select, midi ctl 70
B3	external controlle
B4	foot-controller, midictl
B5	volume, midictl 7
B6	foot switch, sustain pedal, midi ctl 64
B7	pressure (channel pressure
B8	program chang
B9	time comman
BA	sequence call (in song steps)
xx	
BC	end of trac
BD	inst volume

some of the commands given by examples

00-$57, Note messages 6 bytes
These are sufficiently frequent and special that they may deserve specia mention. When parsing: If msbit is set, then check if nybble 1 and 2 (start at 0) is less than than $58. If so it is a note event otherwise not.
The 48. bits in the note message are

Commandbit	:1 bit, always
dClk(hi)	:3 bits, no. of clksX16 to next event(also add dClks(lo)
unused	:1 bit
NoteNumber	:7 bits, note C2 has number $0F , midinote=notenum+33

```
 unused        :4 bits  always 0
 NoteLength    :13. bits (msbit always 0 ), length in units of clks
 unused        :3 bit
 unused        :1 bit   always
 dClk      (lo):4 bits
 Velocity      :7 bits         range 0-127
 extrabit   : 1 bit, the effect of this is not understood? (see footnote 1
 unused        :3 bits always
 ---
 So bit-wise it can be written
  1ttt 0nnn nnnn 0000 0ddd dddd dddd d000 0ttt tvvvv e00
 where t=dClk, n=note, d=duration, v=velocity , e= extrabi
 ( each letter is here a bit, note that other places a single lette
represents a nybble )
 Example ( 90F000A819F0

 Hibit Set=>it is a command,nyb1+nyb2<$58 => it is a notecommand
 NoteNumber=$0F = C
 Length/Duration= $A8 div 8 = $15=21. clock unit
 Velocity=$1F (=31.)=$9F&$7
 extra=0 =ignor
 dClk=$13=$90&$70)+($19 div 8 );  advancement of counter after even


 $58-$AF, polyphonic after-touch 4 bytes
 The note number is found in nybble 1 and 2.  We get the eps note number b
subtracting $58.. ( and  the midi note number by -$58 +33.  )
 8670 0930   note=67-58=0F=ÓC2Ó, dClk=$01(=09 div 8), pressure=$13(=$93&7F

 B0, Pitch wheel 4 byte
   BB00 0410    pitch bend 41=65., dClk=$30 (=(BB&70) + 0
 B1,  Mod wheel 4 byte
   BB10 1740    vibrato $74, dClk=$32(=(BB&70) + (17 div 8) )
 B2, Patch select message 4 byte
  BB20 7C00  patch sel $40=Ò*0Ó, dClk=$3F   (=(BB&70) + (7C div 8)
  CB20 1000  patch sel $00=Ò00Ó, dClk=$42  (=(CB&70) + (10 div 8)
  9B20 3200  patch sel $20=Ó0*Ó(=320&7F), dClk=$16  (=(9B&70) + (32 div 8)
  8B20 47F0  patch sel $7F=Ó**Ó, dClk=$0

 B4, FootPedal 4 byte
   8B40 0410  controlvalue=$41=65. , dClk=$0

 B5, Volume 4 byte
   BB50 03F0 volume=$3F=63. , dClk=$30=48

 B6, FootSwitch 4 byte
   AB60 17F0    switch=$7F=ÓdownÓ, dClk=$22(=(AB&70)+(17 div 8
   AB60 0800    switch= 0=ÓupÓ, dClk=$2

 B8, Program message 4 byte
   BB80 1000    program 0, dClk=$3
 B9, Advance Clock 4 byte
      The rules about  dClk donÕt apply here, the largest dClk is $3FFF
  8B90 01F0    advance clock with $1F=31. ticks,
  9B90 2340    advance clock with $A34=2612.=((9-8)*$800 )+234
 BA, Seq Call  message 8 byte
  BBA0 0080 0000 0020, seq.8 2 times, mute=trans=0 only in songsteps_chunk
  have not seen a 8BA0 variant of this
 BC, EndOfTrack 2 byte
  8BC

 BD,  Inst Volume message 4 byte
  BBD0 0740 , dClk=$30,  inst volume= $7

 Parsing , classi
```

Start parsing at byte offset 28 from top of track. Parse first one intege at a time. If <0 then it is a command (else error). If it is a command then test the second and third nybble (nybble 1 and 2) of the integer. If this is less than $57 its a note -the length=6 bytes, else if less than than $AF then it is after-touch length is 4 bytes else if it is $B0 ... and so forth. One can find the message length from a length table. One then advance the parse pointer accordingly.

SEQUENCES AND THE EPS16

There are many similarities in the sequences of the classic and the 16 sampler. How the sequences are organized is the same. The individual messages are different for the 16+ and the classic however, even if some of the concepts are the same. Generally variables that are integerX16 for the eps classic are integers (with the most significant byte first) for the 16+. Since there are so many similarities we only reproduce briefly the structure and point out the differences. For the 16+ there are also slight differences between single standalone sequences and linked sequences. (There is no such distinction for the classic)

ÒHeader structureÓ 16

The header structure appears very similar to that for the classic, excep the name

Offset	Size	VarName	Meaning/comment
0	4	SeqLen	eps_longin
4	24.	Seq16Name	Name of seq Only the first byte of each integer is used the second is always $FF - except th fourth byte of Seq16Name is 00 for a standalone single seq. (For a songstepseq. the 4th and 6th byte is 00. For a song sequence the msb of each integer is 2
1C	2	NumBars16	This is often 0 for a standalone sequence The numbers can then be calculated fro timesig and the infotrak
1E	2	TimeSig16	time signature, as classic but shift right bit9-12-> nominator, bit 13-15 -> deno denom: 2=quarter,3=eight,4=sixteenth, 1=hal 0=whole example $005C = %0101 1.100 =11/1
20	1	LoopFlg16	off=$0F, on=00
21	1	Tempo16	beats pr. min.
22	36	TrkTbl	9 entries of eps_longints. points to Trk0..Trk

46	Normal start trk0 trk0 = infotr
68	here is the normal start for first recorded trac

ÒSequencer event structureÓ-16

A collection of messages. Each message consist of integers. The number o integers contained in a message is dependent upon what the message is about. The first integer of the message contain information on what kind of command (note, program, mod wheel ..) it is. The length of the message can vary for a given command. See ÕparsingÕ for how to determine if an integer is a command, and if it is, what type it is. The most significant byte is irrelevant for the type of command . byte 1 (the 2nd byte) specifies which command it is. Example: The first integer of the message is $ABCD -here $CD is a byte identifying the command. $A has the most sig. bit set signalling that it is a command. The first byte carries a value $2B(=$AB&7F) which normally is a deltaclock value before next event. For now we ignore the value This means that both BBCD and 80CD would signal the same command. See also below.

Timing of events, 16

The timing information carried in a 16+ sequence has twice the tim resolution of a classic sequence. The 16+ uses 96. clocks pr Q-note while the classic uses 48.. A message contain information on when the next event is to occur. This time information is contained in the most significant byte of the first integer. Example: message 80BD 0022 , Here the command-integer is 80BD. The command byte is BD. The next event will occur after $00 clock-pulses, while for ACBD 0022 the commandbyte is also BD and the next event occur after $2C clock ticks. A track will always start with a time_command to sync the events to the clock. For the 16+ a normal standalone seq. track usually starts with Òcount down 3 clock pulsesÓ before next event, while the classic starts with 1 clock pulse as does a 16+ sequence within a song. A standalone 16+ sequence can also start with count down 1 clock pulses. Trk0 which we also calls the infotrack contains the number of clock ticks in the sequence, from this number and the time-signature one can calculate how many bars there are. Note that normally the number of ticks in a 16+ sequence will be twice that in a classic plus one, given the same input for standalone sequences. For a song sequence the number of ticks in the info track will be exactly the number of tics as calculated from the time-signature and the number of bars (not + 1)

The different commands, 16+

As noted above the second byte (byte 1, $CD in $ABCD) contain the key t identifying the command. First we present a list of these commands, then we list the different values for these commands, indicate how many bytes a message of that type will contain and give an example of a message and decode it.

List of command

00-57 note message, short: 80nn v[v+d]dd, where each entity is a nybble
nn=notenumber, v=velocity, d=duratio
ex 8027 1F23, note=27, vel=1C, dur =323
long : 8027 0800 112
58-AF after-touch, 8069 0035, note=11(=69-58), afttch=3
B0 pitch wheel, 80B0 00pp
B1 modulation wheel 80B1 00m
B2 patch select , 80B2 00XX, where XX is patch as for classi
B3 external controlle
B4 foot-controller, midictl 4
B5 volume (pedal), midictl 7 , 80B5 00v
B6 foot switch, sustain pedal, 80B6 007F, 80B6 000
B7 pressure
D9 program change(classic B8), 80D9 00pp
DB pan , 80DB 00p
DD inst load from disk, 80DD 00i

E6 time command (classic B9), 80E6 tttt, msbyte probably X800
E7 sequence call (in song steps)(classic BA)
XX
E9 end of track (classic BC
DA Volume, trackvolume (classic BD), 80DA 00v

Note comman
The note command integer can be followed by 1 or 2 data integers. If th duration of the first data integer is zero (bit6-bit15=0) then there are two data integers, else one. The duration is kept in bit6 to bit15 of the first data-integer if there is one integer. If there are two data integers then the duration is kept in the second integer read as a normal Motorola type integer- most sig. byte first. The velocity is kept in bit0-bit5 of the first byte of the first data integer. The range of the velocity is 0-127,

but the resolution is 4,so the values are 0,4,8,C,10,14,18,1C........ The 3 integer variant is used when the duration is longer than $3FF clks.(This indicate that the longest possible note is $7FFF (=85 bars of 4/4)

Examples

8027 013C   : note=27, vel=0(=midi3), dur=13C, dClks=0

9029 0800 0764 : note=29, vel=8(=08&7C), dur=764, dClks=10(=90-80

Time comman

I have not tested if longer delays than $7FFF are possible, and if so wha are the rules. If the rules are similar to that on the classic we use the excess in the msbyte as a multiplier of $8000 yielding a max. delay of $3FFFFF. Example:

92E6 1523 dClk= 91523=(92-80)*8000 +152

Parsin

Start parsing at byte offset 28. from top of track. Parse first one intege at a time. If <0 it is a command integer (otherwise error). Advance the parser by as many integers as required by the number of data integers for the command. They should be positive, else error.

ENSONIQ EPS SONG STRUCTUR

A Song consist of a collection of individual (but stored together in file) sequences, one sequence for the song and a special songsteps chunk.

EPS classic and 16

The song file start with a collection of pointers to the individua sequence chunks. The number of sequences does not seem to be stored so one would have to count. The Song Sequence has a pointer at a fixed position.

SONG HEADER, classic and 16

The header consist of mostly vectors to sequences. All data are stored a eps_longs. (example $12300040 points to $4123) . At location $150 is a pointer to Òthe songs tracksÓ-here called song sequence. Normally the Òsong step sequence chunkÓ will follow after, (but may not and)the entry point is anyhow included in the list of sequence chunk pointers. One can also tell what is a normal sequence and what is a song-step sequence by looking at the name field in the sequence chunk. For the classic if byte 20 is $30 its a normal sequence if it is 0 its a song step sequence. For the 16+ if it is a normal sequence the sixth byte is FF if it is 00 it is a song step sequence.

Song_chunk, classic and 16

ofs	size	name	meanin
0	4	SongSz	512 bit chunky size of song stored as eps_lon
4	4	NextSeq	physical size of song stored as eps_longin
8	4	songstepPtr	Pointer to the songstep chunk, eps_longin
C	$134	SeqTbl	offsets stored as an array of eps_long_pointers, empty slots are filled with 0. The sequence are numbered by order of appearance in the tabl starting with 1. (seq0 is the SongSteps)
$150	4	SongSeqPtr	offset (eps_long_pointer) to the song sequence.

then comes all the sequence_chunks in an arbitrary order.

Song_steps_chunk, classi

ofs	size	name	meaning/comment
0	4	SeqLen	eps_longin
4	24.	SeqName	msb of each integer filled with ascii, ofs 20=
28.	2	NumSongSteps	stored as integerX16 ÒORÓ RepFlag(on=0800

```
 30.  2     songnum?   x16, usually zer
 32.  var.   SongSteps  an array of individual commands,
                       these are normally 8BA0(= seq. call ), and eotr

 seq._call_command, classi

 $8BA0 0080 0000 0000 0020
 means at current time call sequence 8 twice,
 written as bit
 <1stint>| <trk>|{0} mmmm mmmm {0} | {0} tttt tttt {0}| 0nnn nnxx xxxx {0}
 here m=mutebyte, t=transposebyte,n=transposeamount, x=repeattimes
 {0} means a nybble where all bits are 0
 the mute and transpose byte has track 1 to the right and track 8 to th
left,
 example:$8BA0   0040 0220 0050 3060 will play sequence 4 repeated 6 times
tracks 1 and 3 will be transposed +12 steps, track 2 and 6 will be muted.
The transpose bits is a 5 bit signed 2Õs compliment number
 %10100=-12   %10101=-11   %10110=-10   %10111=-
 %11000=-8    %11001=-7    %11010=-6    %11011=-
 %11100=-4    %11101=-3    %11110=-2    %11111=-
 00000=0      %00001=

 Song_steps_chunk, 16

 ofs  sz        name        meaning/comment
 0    4         SeqLen      stored as eps_longin
 4   24.        SeqName16   msb of each integer filled with ascii ,
                            lsb=FF, except ofs 4=ofs 6 =
 28.  2        NumSongSteps max. 99, 'OR'ed with Rep Flag (=$0080) ???
 30.  2         SongNumber? please try!!!!!!. itÕs normally
 32. var.        SongSteps   an array of individual commands,
                            these are normally  80E7, (= seq. call
                            ended by EndOfTr
 seq._call_command, 16

 $80E7 0001 0000 0000 0001
 means at current time call sequence 1 once,
 written divided in bytes by {} and in integers by |
 {80}{E7}|{0}{trk}|{0}{mutetrk}|{0}{transp.trk.}| {trans.v}{rep}
 here trk=track, transp.trk tracks to transpose, trans.v the amount t
transpose, rep= how many times to repeat. To get the transpose amount, you
shift right 2 times and sign extend the byte from the second bit.
 The mutetrk and transp.trk bytes uses each bit as a mask; track 1 to th
right and track 8 to the left,
 example:$80E7   0004 0022 0005 3006 will play sequence4 repeated 6 times
tracks 1 and 3 will be transposed +12 steps, track 2 and 6 will be muted.
The transpose byte yields
 $50=-12.   $54=-11.   $58=-10.  $5C=-
 $60=-8     $64=-7     $68=-6    $6C=-
 $70=-4     $74=-3     $78=-2    $7C=-
 $00=0      $04=1      $08=2     $0C=
 $10=4      $14=5      $18=6     $1C=
 $20=8      $24=9      $28=10.   $2C=11.   $30=12
 ( where the left of the Ò=Ó sign is the transp.v byte and the right is th
transpose amount in half steps )


 FOOTNOTE
 footnote 1  : About One bit not understood in the note command.
 One could think this is pretty serious, and maybe it is.  Often this bit i
zero.  I have seen it been 1. This is not randomly. If this bit is 1 it is
so for all note commands for a take.  If I change this bit (by editing the
file), the sequence still plays back, and I cannot hear any differences.
That sort of tells me I can get by without having to worry about it.
```

footnote 2 : References to eps-file informatio

1) Articles written by Gary Giebler in Transoniq Hacker.
   Supposedly appeared in autumn 1991.
2) The essential parts of the above was edited by Scott Fisher and
   sent to the eps-mailing list {<m017GDM-0003HFC@reed.edu>,
   24 Jan 92 in the EPS mail-archive at eps.reed.edu }.
3) A copy(?) of 1) is in a document called EPSDiskFormat.Z (sp?)
   that can be found at eps.reed.edu.
4) A way of storing individual files that has been adopted by
   Gary Gieblers utilities for the eps and PC as well as
   Steve Quartlys utilities for the eps and Atari is called the .EF
   format. This file format is explained by Steve Quartly
   {Letter from Steve Quartly,<m01cUUk-0003G1wC@reed.edu>
   18/19 Apr. 92 in the EPS mail-archive at eps.reed.edu}.
5) You can also find all about the file structure as well as all the
   info contained in this document by peeking around the disk yourself.
6) If you want to read images you may want info on the .gkh file forma
   used by the eps-mailing list. This is described by Goh King Wh
   { Letter from Goh King Wha, <m01T8r50003FwC@reed.edu>, 24 Mar 92,
   in the EPS mail-archive at eps.reed.edu
7) Also for C code for reading EPS disk directories se
   Letter from Kelly Larson <m0lq9nZ-0003HVC@reed.edu>,26 May 1992 in
   the EPS mail-archive at eps.reed.edu and utilities at eps.reed.ed

---

The Information found is this document is intended to be useful an correct, however no such guarantee is given and any use of the information is on the users own risk. Any corrections or additions to this manuscript is welcomed . (contact <t.g.finstad@fys.uio.no> by e-mail)