

ENSONIQ EPS/ASR 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 to follow the text.

The ASR-10 sequence is very similar to those for the EPS16+ for sequences and songs without audio tracks. So the ASR-10 sequence do not need a separate description. The ASR-10 will read EPS Classic sequences and convert them to ASR sequences. However there are very many bugs in that conversion.

Sequences can be stored individually on disk, or several sequences can be 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 faint similarities with the other sequences and that is the SONG STEPS. Note that the differences between a Standalone-, linked-, and Song-sequences are small and may not have been intentional.

In order to take direct advantage of the information that is given here you first have to access Ensoniq 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 notations

Some numbers found in eps sequences may at first appear strange when viewed 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 "12 bit wide memory" 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. The EPS has a 13 bit wide memory, but that's beyond the point. Here we describe one common structure 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 we 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 "eps_longint."

When I in this document write \$1234 I use "\$" to indicate hexadecimal, I use %110011 to indicate binary and I use 20. (note the decimal point) 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. I'll use common assembler symbols for logical operations on numbers, example & is the symbol for logical and.

"Header structure", classic

We start with a table which is a listing of the start of a sequence file, which is the header of the sequence.
(in the tables the sizes and offsets (ofs) are given in bytes unless otherwise specified,)

Ofs	Size	VarName	Meaning/Comments
0	4	SeqLen	Length of sequence in bytes stored as eps_longint
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 sequence (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 denom denom: 2=quarter,3=eight,4=sixteenth, 1=half 0=whole example \$05C0 = %1011 1.100 0000 =11/16
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_long
22	4X9	StartTrkTbl	
26	4	StartTrk1	Start address from top of file track1
2A	4	StartTrk2	-----" -----2
2E	4	StartTrk3	-----" -----3
32	4	StartTrk4	-----" -----4
36	4	StartTrk5	-----" -----5
3A	4	StartTrk6	-----" -----6
3E	4	StartTrk7	-----" -----7
42	4	StartTrk8	-----" -----8
46			here is the normal start for infotrak = trk0
68			here is the normal start for first recorded track

Track_Chunk structure classic and 16 plus

In a sequence there appear to be possible to have 9 Track Chunks all having 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 significant 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/comments
0	4	TrkLength	Byte length of Track stored in an eps_long
4	4	unknown	, set it to 0
8	20.	Scratch	Unknown, Not used when read/writing to floppy
\$1C	..	eventstart	A sequence of messages, ended by the EndOfTrack command.

"Sequencer event structure"-classic

It may be appropriate to regard the sequencer events as a collection of 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 'parsing' 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, classic

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.. It's 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, classic

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 commands

00-57 note message
58-AF after-touch, key pressure
B0 pitch wheel, midi pitch bend
B1 modulation wheel, midi ctl 1
B2 patch select, midi ctl 70.
B3 external controller
B4 foot-controller, midictl 4
B5 volume, midictl 7
B6 foot switch, sustain pedal, midi ctl 64.
B7 pressure (channel pressure)
B8 program change
B9 time command
BA sequence call (in song steps)
xx
BC end of track
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 special 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 1
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 bits
unused :1 bit always 0
dClk (lo):4 bits,
Velocity :7 bits range 0-127.
extrabit : 1 bit, ignore this bit. see footnote 1.
unused :3 bits always 0

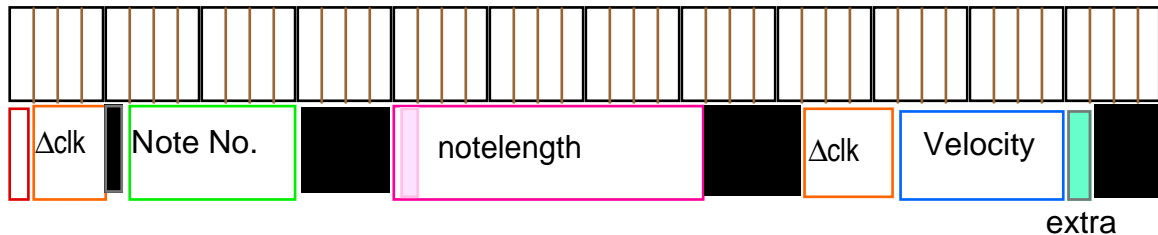
So bit-wise it can be written:

1ttt 0nnn nnnn 0000 0ddd dddd dddd d000 0ttt tvvvv e000
where t=dClk, n=note, d=duration, v=velocity , e= extrabit
(each letter is here a bit, note that other places a single letter
represents a nybble)

We illustrate an example below

Example command in bytestream (90F000A819F0)

9 | 0 F | 0 0 0 A 8 | 1 | 9 F | 0



This is decoded in the following way:

Hibit is set => it is a command, nyb1+nyb2<\$58 => it is a note command.
 NoteNumber=\$0F = C2
 Length/Duration= \$A8 div 8 = \$15=21. clock units
 Velocity=\$1F (=31.)=\$9F&\$7F
 extra=0 =ignore
 dClk=\$13=\$90&\$70)+(\$19 div 8); advancement of counter after event

\$58-\$AF, polyphonic after-touch 4 bytes:

The note number is found in nybble 1 and 2. We get the eps note number by subtracting \$58.. (and the midi note number by -\$58 +33.)
 8670 0930 note=67-58=0F="C2", dClk=\$01(=09 div 8), pressure=\$13(=\$93&7F)

B0, Pitch wheel4 bytes

BB00 0410 pitch bend 41=65., dClk=\$30 (=(BB&70) + 0)

B1, Mod wheel4 bytes

BB10 1740 vibrato \$74, dClk=\$32(=(BB&70) + (17 div 8))

B2, patch select message4 bytes

BB20 7C00 patch sel \$40="*0", dClk=\$3F (=(BB&70) + (7C div 8))

CB20 1000 patch sel \$00="00", dClk=\$42 (=(CB&70) + (10 div 8))

9B20 3200 patch sel \$20="0*" (=320&7F), dClk=\$16 (=(9B&70) + (32 div 8))

8B20 47F0 patch sel \$7F="***", dClk=\$08

B4, FootPedal4 bytes

8B40 0410 controlvalue=\$41=65. , dClk=\$00

B5, Volume4 bytes

BB50 03F0 volume=\$3F=63. , dClk=\$30=48.

B6, FootSwitch4 bytes

AB60 17F0 switch=\$7F="down", dClk=\$22(=(AB&70)+(17 div 8))

AB60 0800 switch= 0="up", dClk=\$21

B8, Program message4 bytes

BB80 1000 program 0, dClk=\$32

B9, Advance Clock4 bytes

The rules about dClk don't 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 bytes

BBA0 0080 0000 0020, seq.8 2 times, mute=trans=0 only in songsteps_chunks have not seen a 8BA0 variant of this

BC, EndOfTrack 2 bytes

8BC0

BD, Inst Volume message 4 bytes

BBD0 0740 , dClk=\$30, inst volume= \$74

Parsing , classic

Start parsing at byte offset 28 from top of track. Parse first one integer 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 \$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)

"Header structure" 16+

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

Offset	Size	VarName	Meaning/comments
0	4	SeqLen	eps_longint
4	24.	Seq16Name	Name of seq. Only the first byte of each integer is used the second is always \$FF - except the 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 2E
1C	2	NumBars16	This is often 0 for a standalone sequence. The numbers can then be calculated from timesig and the infotrak.
1E	2	TimeSig16	time signature, as classic but shift right 4 bit9-12-> nominator, bit 13-15 -> denom denom: 2=quarter,3=eight,4=sixteenth, 1=half
			0=whole example \$005C = %0101 1.100 =11/16
20	1	LoopFlg16	off=\$0F, on=00
21	1	Tempo16	beats pr. min.
22	36	TrkTbl	9 entries of eps_longints. points to Trk0..Trk8
46		Normal start trk0	trk0 = infotrak
68		here is the normal start for first recorded track	

"Sequencer event structure"-16+

A collection of messages. Each message consist of integers. The number of 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 'parsing' 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 delta clock 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 time 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 "count down 3 clock pulses" 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 to 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 commands

```

00-57 note message, short: 80nn v[v+d]dd, where each entity is a nybble
                        nn=notenumber, v=velocity, d=duration
                        ex 8027 1F23, note=27, vel=1C, dur =323
                        long : 8027 0800 1123
58-AF after-touch, 8069 0035, note=11(=69-58), afttch=35
B0 pitch wheel, 80B0 00pp
B1 modulation wheel 80B1 00mm
B2 patch select , 80B2 00XX, where XX is patch as for classic
B3 external controller
B4 foot-controller, midictl 4
B5 volume (pedal), midictl 7 , 80B5 00vv
B6 foot switch, sustain pedal, 80B6 007F, 80B6 0000
B7 pressure
D9 program change(classic B8 ), 80D9 00pp,
DB pan , 80DB 00pp
DD inst load from disk, 80DD 00ii

E6 time command (classic B9 ), 80E6 tttt, msbyte probably X8000
E7 sequence call (in song steps)(classic BA )

```

xx
E9 end of track (classic BC)
DA Volume, trackvolume (classic BD), 80DA 00vv

Note command

The note command integer can be followed by 1 or 2 data integers. If the 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 command

I have not tested if longer delays than \$7FFF are possible, and if so what 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 \$3FFFFFF. Example:

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

Parsing

Start parsing at byte offset 28. from top of track. Parse first one integer 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 STRUCTURE

A Song consist of a collection of individual (but stored together in 1 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 individual 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 as eps_longs. (example \$12300040 points to \$4123) . At location \$150 is a pointer to "the songs tracks"-here called song sequence. Normally the "song step sequence chunk" 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	meaning
0	4	SongSz	512 bit chunky size of song stored as eps_long
4	4	NextSeq	physical size of song stored as eps_longint
8	4	songstepPtr	Pointer to the songstep chunk, eps_longint
C	\$134	SeqTbl	offsets stored as an array of eps_long_pointers, empty slots are filled with 0. The sequences are numbered by order of appearance in the table 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, classic

ofs	size	name	meaning/comments
0	4	SeqLen	eps_longint
4	24.	SeqName	msb of each integer filled with ascii, ofs 20=0
28.	2	NumSongSteps	stored as integerX16 "OR" RepFlag(on=0800)
30.	2	songnum?	x16, usually zero
32.	var.	SongSteps	an array of individual commands, these are normally 8BA0(= seq. call), and eotrk

seq. call command, classic

\$8BA0 0080 0000 0000 0020

means at current time call sequence 8 twice,
written as bits

<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 the 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 2's compliment number

%10100=-12 %10101=-11 %10110=-10 %10111=-9

%11000=-8 %11001=-7 %11010=-6 %11011=-5

%11100=-4 %11101=-3 %11110=-2 %11111=-1

00000=0 %00001=1

Song_steps_chunk, 16+

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

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 to 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 the 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=-9
\$60=-8 \$64=-7 \$68=-6 \$6C=-5
\$70=-4 \$74=-3 \$78=-2 \$7C=-1
\$00=0 \$04=1 \$08=2 \$0C=3
\$10=4 \$14=5 \$18=6 \$1C=7
\$20=8 \$24=9 \$28=10. \$2C=11. \$30=12.

(where the left of the "=" sign is the transp.v byte and the right is the transpose amount in half steps)

FOOTNOTES

footnote 1 : About One bit extra in the note command.

Often this bit is 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 the same. I have concluded that one do not have to worry about it.

footnote 2 : References to eps-file information

- 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 oak.oakland.edu.
- 4) A way of storing individual files that is used by utilities sold by Gary Gieblers for the EPS and PC is called the .EFE format and is also a trademark.
- 5) You can also find all about the file structure by peeking around the disk yourself.
- 6) You can read images (of the floppy)which holds the sequence files by a number of existing utilities, and you can easily make your own too.
- 7) For C code for reading EPS disk directories see Letter from Kelly Larson <m01q9nZ-0003HVC@reed.edu>,26 May 1992 in the EPS mail-archive at eps.reed.edu and utilities at eps.reed.edu

The Information found in this document is intended to be useful and 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 <terje.finstad@fys.uio.no> by e-mail)