

Simple FAQ on Ensoniq related file format specifications

Mostly for programmers or other interested geeks. Not aimed at software users and abusers.
The selection in this note is very much PC related.

There are no official format specifications of the formats described here. EFE , EFA, EDE and EDA are trademarks by Giebler Enterprises who use the format names in their utilities. GKH is a file format named by the EPS mailing list in 1992 to describe the files made by one particular program (written by Goh King Wha).

For a programmer who will write a new utility, it is advantageous to know the existing formats. It is an advantage to the programmer, but really mostly to any users of his/hers utility and an advantage to users and supporters of existing utilities. Writing utilities that take a private approach to existing formats will create problems for many. The point is already proven. Maybe this note helps.

Q: Any clues about those *.EFE and *.EFA file formats?

A: Gary Giebler from Giebler Enterprises made those formats and uses them successfully for what they were intended for with their utilities on the PC. Thus, it is that company that defines those particular formats. In order to stay on top of any changes or evolutions you can buy the latest version of the program EDM from Giebler when a new comes out and make test files that you can examine to find out about the format and/or of changes. It can in principle change at any time, but since there appears to be no new work going on adapting the old Ensoniq formats in new products, it is likely that there will be no more changes to the format. The following description is based mostly upon the files made by various versions of EDM which you can find around the Internet.

Format wise, EFE and EFA is the same thing. It is just Ensoniq single files (Instruments, Banks, Effects etc.) for the EPS, 16+ and ASR respectively. You can extrapolate for VFX and TS10/12. (EFV and EFT) The *.EF? file has a 512 byte header in front of the copy of the data file as found on an Ensoniq floppy. The header serves two purposes: 1) As a service to DOS users it contains some text information about the content of the file. 2) Giving information on the format and it's content to programs reading the file. The latter contains critical marks at particular locations which can be use for it's identification. In addition it contains some information/data which on an Ensoniq floppy is contained in a directory. On an Ensoniq floppy, each file has an entry in a directory (database) on a floppy. That entry tells how big the file is, what kind of file it is, it's name and few other things. This information is copied to the header.

The EFE/EFA format was made with MS-DOS in mind. At the DOS command line machine you can write 'type bass.efe' to see what the file "bass.efe" consists of. (- Is it an EPS file? Is it an instrument? or a sequence?) The DOS system will then start typing the header. It will stop typing when it encounters the IBM end of file character, which will occur in the header. You will see the details of what is typed below from the structure of the header. In the header of an EFE file there are some non essential text which is only for the users information. Then there is essential identifying content; The carriage return (0xD) and end of file marks (0x1A) are at fixed positions in the header and are essential. They can be used in identifying the format. Part of the directory entry of the file is also in the header. Unfortunately there has been some evolution there. Giebler did not just copy the directory, which would have been safer with respect to future compatibility , but as stated above, that is of no concern anymore (You see!, this was written first in 1992, Then it was. a concern I've updated the document! Now in 2000, it is of no more concern) Giebler copies single bytes or numbers from the directory and puts them at different locations in the header.

From the directory entry (found on the Ensoniq floppy), you need the file name, the file type and the block length of the file. This is essential information put in the header but not used for identification (other than for distinguishing instruments from sequences)

Here are the offsets with their contents in the header

```
0   : 0x0D   ; essential
1   : 0x0A   ; essential
2   : 'EPS file:' Non essential, can be 'Eps file', 'ASR file' or anything.
```

0x12: File Name 12 ascii upper case chars from the directory entry
0x22: File type name, non essential, for example 'Bank'
0x2F: 0x0D ; essential
0x30: 0x0A ; essential
0x31: 0x1A ; essential, IBM end of file
0x32: File type; Byte, Essential, for example 3 for Instrument
0x34: Block File Size: Long[4 bytes] Motorola ordering, Essential
0x38: Ptr First Block: Integer[2 bytes]Motorola ordering
0x3A: multi File index: byte usually 0

In addition to these I use (in EPSm and stEPS)

0x3E: 'EFE1' ; my signature
0x42: A direct copy of the directory entry for the file, 26 bytes
I use the data in this directory if 'EFE1' is present,
otherwise I assemble the directory based on the entries above.
Don't put 'EFE1' in a file unless you also make a direct copy of
the 26 bytes in the directory.

For file type byte the following list put the file types in order

```
( {  
  BLNK, OS, SUB, INST, BANK, SEQ, SONG, SYSX, PPTR, MACR, {  
  sd1_1, sd1_2, sd1_3, sd1_4, sd1_5, sd1_6, sd1_7, sd1_8, sd1_9, {  
  sd1_10, sd1_11, sd1_12, sd1_13, {  
  LK16, EFF, SQ16, SO16, OS16, {  
  SEQa, SNGa, BNKa, ATrk, OSa, EFFa, MACa, {  
  pg6, pg60, p120, ps1, ps60, SetUp, seq1, so30, demo, eu44, cnfg, smpbnk,  
EDIT  
  );
```

Here BLNK means blank and has 0 as filetype number

INST means Instrument and has number 3

You can of coarse call them what you want, In the list, sd1_10 to sd1_13
are SD files, LK16 to OS16 are 16+ files (LK16 means Bank (Link))
SEQa to MACa are for the ASR and pg6 to EDIT are for the TS10/12.
At least some VFX files fall in the range set off for SD above.

When writing a new file I would fill the header with blank space up to offset
0x2E before I filled in the other text parts, essential markers and the
essential data.

Note that there are utilities that do not adhere to these recommendations and advices.

Some utilities will treat the text at offset 2 as text unique to the format. That is a bad idea
since there are several utilities that write the text differently. (and EDM does not check the spelling).

Further there are utilities that forget to put some of the info from the directory into the header.

There are also utilities that produce garbage EFE files unless they originate from floppies
with no fragmentation. The only way to test for this is to test each kind of file in a unique way.

The last block of every file type should adhere to the way that filetype should end. You need
the structure of every file type to cope with this, but some types, like Instruments would be dominating
and should thus be the most important to check .

Q: Any clues about those *.EDE and *.EDA file formats?

A: EDE and all Gieblers ED* are mostly the same thing. The format has a 512 byte header,. Again the carriage returns and

EOF-marks are essential. The header also contains a bit map telling which blocks from a floppy are in the file. These blocks follow the header. If the bit in the bit map is 0, the block is in the file. The bit map starts at different addresses for HD and DD floppies. Whether it is an EDE, EDA, EDT or EDV file is contained in a number at offset 0x1FF,-see the blank files that comes with the EDE107 package shareware of Giebler for the identification of all the Ensoniq synths.

The fixed markings are at..

```
00:    0x0D;
01:    0x0A;
0x4E :0x0D
0x4F :0x0A
```

For HD floppies the bitmap start is at 0x60.

For DD floppies it is at 0xA0.

The marking just before the bitmap should be 0x0D0A1A

For EDE offset 0x1FF is 3

For EDA offset 0x1FF is 0xCB and signals a HD floppy

For EDT offset 0x1FF is 0xCC for HD floppies

For EDT offset 0x1FF is 0x07 for DD floppies

For so called COMPUTER FORMAT/size disks offset 0x1FE is 1 otherwise 0.

It can be used for EDE and EDA but no synth can use the size for TS data.

Q: Any clues about those *.GKH files?

A: GKH is what the PC programs EPSREAD.EXE and EPSWRITE.EXE makes; nothing more, nothing less!!

GKH should contain an image of an 800 kb DD Ensoniq sampler floppy. It should have a header that tells it is a GKH file. The header should thus be like EPSRead makes the header.

Unfortunately, this practice has not been followed by all program authors, so some users may naturally have gotten used to something else. This concern in particular making a private extension such as done by EPSDisk for other floppies and naming the file *.gkh. There is no good reason for this however.

The program EPSDisk lets you make a raw image of any kind of floppy. You can read and write any floppy size and kind by using the file extension .IMG and that is what you should use for images of other sizes than 800 kb. Please see the description of EPSDisk. Don't use GKH for anything but 800 kb floppies in public.

It has become common to confuse the GKH format with thoughts laying behind the creating program.

GKH is fixed. That ensures the largest degree of compatibility. There are no need for expansions.

GKH is what the PC programs EPSREAD.EXE and EPSWRITE.EXE makes; nothing more, nothing less!!

Here is a description of the header of a GKH file, I describe the bytes first as they appear from offset 0.

```
0x00: FileIndicator - four bytes Always 'TDDF'
0x04: NumberTypeIndicator - ascii always 'I',
      ( This signals Intel, -PC type, byte ordering
        for numbers, a 'M' for Motorola- Mac is NOT used )
0x05: VersionNumber - byte always 01
0x06: NumberOfTags - word ( 2 bytes Intel ordering )
```

Then follows a variable number of Tags as indicated by the above. It is recommended that there are 5 Tags.

Each Tag is 10 bytes. They give various pieces of information on the contents of the file. They tell what is in the file and where it is. It is recommended that the image starts immediately after a 58 byte header (5 Tags) but every program reading a gkh file should check the tag fields to check that so is the case. The tags can come in any order. The minimum number of Tags for GKH0 is 3. These are called FormatTag, ImageTypeTag, ImageLocationTag. The optional ones are AuthorTag and SubjectTag. The FormatTag and ImageTypeTag will always be constant for GKH0.

```
FormatTag      - - - - $010401000000001000000
ImageTypeTag   - - - - $0A05500002000A000002
ImageLocationTag - - $0B0B00800F00xxxxxxx where xxxxxxxx is the offset to the image
AuthorTag      - - - - $140ALLLLLLLLlyyyyyyyy where yyyyyyyy is the offset to the image
SubjectTag     - - - - $150Asssssssszzzzzzzz where zzzzzzzz is the offset to the image
```

In the above, xxxxxxxx is normally 3A000000 which is offset 58, NB word aligned. LLLLLLLL is the length of the author info ssssssss is the length of the subject info

The history behind the tags can be ignored, since they are completely defined by the above, but it is OK to know when parsing them. So here is the historic perspective on the structure of tags:

```
first byte- tag type
2nd byte - organization of the remaining 8 bytes ( 8 bytes, 2 longs, or 4 shorts or..)
the following values are defined:
0B - next comes two longs
0A - next comes two longs
05 - next comes 4 words/shorts
04 - next is two longs
byte 2-9 - depends on tag type ( and byte ordering )
```

The FormatTag is a constant, so one does not need to know the ancient perspective, but the two longs are: 01 for EPS Disk, 01 for data dump type which is uncompressed. The ImageTypeTag is also a constant. The words are 0x50 for 80 tracks, 02 for two heads, \$0A for ten tracks 0x200 for 512 bytes per block. The first long of the ImageLocationTag is also a constant which holds the length of the image which is. 10*2*80*512. Again, for GKH these numbers are fixed. For some other format they can be anything.

When an application reads a gkh file it may check first the FormatTag and the ImageTypeTag. Then the application might want to check that the image which is found by the ImageLocationTag in fact is an image of the proper type, i.e. check that it is an Ensoniq floppy, check that the number of tracks is correct and so on. This can be done by checking specific bytes in the device block of the image. Further checks should include the last bytes of the directory block and the last bytes of the FAT blocks.

Q: Any clues about those *.IMG file formats for Ensoniqs?

A: This is the simplest and safest format you can think of and the most easy to implement in a program. It consist of a block by block direct copy of a floppy.

Q: Any clues about EPS, +16, ASR sequences?

A: You can look at a [dedicated document. on the sequence and song formats for EPS, 16+](http://fysmac-elg01.uio.no/eps/EPSfileFormatsFAQ.html) (and ASR)

Ensoniq File descriptions

It contains all you need to convert the Ensoniq Formats to/from Midi - except the Midi info :-), but the latter can be found many places on the net

Terje