

The Experimental Fat Loader (EFL) © Phil Simmons 2007

Licence – use EFL for whatever you like but please credit me. If you screw up and wipe your data - it's your own fault not mine ☺

What is EFL?

EFL is an experimental boot loader that will load a disk file into memory and then execute it. The file is stored using a File Allocation Table (F.A.T), similar to the system used by MS-DOS.

However the EFL is substantially simplified and more efficient than the Microsoft offering-

1. It abandons the MS use of FAT12 for floppy disks and uses FAT16
2. It can (and will) be easily adapted to FAT32.
3. The boot parameter block is different and more compact.
4. Clusters 0 & 1 are not reserved by the system & only 1 copy of the FAT is used

Why use EFL?

I wrote EFL to illustrate several things

1. There's been much forum discussion about the advantages of F.A.T over the CP/M Allocation Block system but seeing and playing is the only way to gain a real understanding of Fats' true potential.

2. The XTAL OS is booted straight from the first disk track.

All Einstein boot disks start with 3 word values. These 6 bytes, written at track 0, sector 0, determine what is loaded.

The 1st word is the Load Start address – where in memory the code begins. This must be on a 256 byte page boundary (eg. 100H).

The 2nd word is the Load End address – and determines how many sectors are loaded – the address is filled to the next 256 byte page boundary.

Finally, the 3rd word is the automatic execution address or entry point for the code.

When the Einstein first cold boots a disk, it reads the first 2 sectors into the system sector buffer at FE00H. The 2 sectors are then copied to the load start address.

If this not given, i.e. = 00, then it defaults to 100H. If the Load End address is greater than 2 sectors then the memory is filled accordingly. Execution takes place at the entry point specified. The minimum boot is 2 sectors.

Also, code cannot be executed below 100H because MOS sweeps this area on initialisation. All these features mean it's easy to boot anything on the Einstein, without having to read a file in first. The down side is that this tends to make an OS dependent on the hardware (e.g. size of FDD), leads to a very large boot area on a disk, and makes the OS inflexible in that it's difficult to use things like overlaid code segments to add functionality while saving space.

3. EFL boots a file (see the source). The file is stored as a FAT entry (see 4). The file can contain an OS loader, OS code and can be used as resource for overlaid code segments, error messages etc.

4. A directory entry need only contain the number of the first cluster the file uses. This has several advantages. In CP/M all a files' pointers to allocated blocks are stored in the directory entry. This means that a directory entry can rapidly become full. Another extent of the file

must be created to accommodate additional allocation blocks. This consumes directory space and there is a limit to the number of extents that can be opened, thus limiting file size. With FAT these problems are avoided.

How to Use EFL

The asm code should be assembled and written to a disk. An initial XCOMMAND.BIN file has been created, comprising 4 sectors. The file contains a demo sign on message. Simply paste your code into the file image section of the source code marked, DATA section. If you need more room extend the allocated clusters in the FAT manually. The last cluster of a file must end in \$FFFF to denote the end of the chain.

The asm source was created under WindozeXP, using the Context programmers editor, assembled and transferred to the Einstein using EDIP and CPCDISK1.6

A pre-assembled object code file is provided on an XTal205 40 track DSK image. Create an Einy boot disk using CCPDISK.

Once booted, simply LOAD the EINFAT.OBJ code (26 blocks), drop into MOS and do a W100 1AFF onto a new disk.

Boot the new disk and the file will load and execute.

What EFL isn't & what it can do

Don't expect EFL to run CP/M programs – duh, there isn't an OS loaded! – you could write one and put it into the boot file – that's what it's for ☺

EFL is squashed into a small can – actually it's not easy to fit a boot loader into 512 bytes especially in Z80. Intel X86 op codes do a lot more work in single bytes since they're true 16 bit – so it's been a squeeze.

The loader hasn't been exhaustively tested and I can't guarantee its bug free. Optimisation is for space not speed and any code improvements or suggestions are welcome.

I'm happy of course to discuss EFL anytime on the forum or directly by e-

mail, killspam<phil_simmons@bluebottle.com>killspam

Can you use EFL to directly read an MS formatted disk? – Yep, if can you be bothered to modify for FAT12 and other MS nonsense, I can't.

Since EFL uses LBA mode it's ideal as test bed to write a file manager for an LBA addressable device like a Compact Flash or HDD.

I'm using EFL to write my FX80 DOS system for the Einstein. I won't go into much detail here suffice to say that it should support huge disks, long file names, paths and unlimited directory entries and should run 99.99% of CP/M programs etc, .etc., but yeah, it will run off a floppy too ☺

Will there be updates and improvements to EFL? – Probably, as the need arises, but remember EFL has been made as simple as possible to make it a flexible tool for you to modify for your own purposes.

Files in EINFAT.ZIP

EFLDOC.PDF	...	This documentation
EFLREAD.ME	...	CP/M text file
EINFAT.ASM	...	The source code
EINFAT.OBJ	...	Einstein binary file
XTAL205TRANS.DSK	...	40 track XTAL DOS 2.05 image & copy of EINFAT.OBJ on disk

EFL HELP

The source code is fully documented and careful reading should answer most of your questions. But if you're stuck, please post the forum.

Happy programming and enjoy!