EFSL

Embedded Filesystems Library - 0.3

Lennart Yseboodt
Michael De Nil

Contents

1 Preface

1.1 Project aims

The EFSL project aims to create a library for filesystems, to be used on various embedded systems. Currently we support the Microsoft FAT filesystem family. It is our intention to create pure ANSI C code that compiles on anything that bears the name 'C compiler'. We don't make assumptions about endianness or how the memory alignment is arranged on your architecture.

Adding code for your specific hardware is straightforward, just add code that fetches or writes a 512 byte sector, and the library will do the rest. Existing code can be used, writing your own code is only required when you have hardware for which no target exists.

1.2 Project status

Efsl currently supports FAT12, FAT16 and FAT32. Read and write has been tested and is stable. Efsl runs on PC (GNU/Linux, development environment), TMS C6000 DSP's from Texas instruments, and ATMega's from Atmel. You can use this code with as little as 1 kilobyte RAM, however if you have more at your disposal, an infinite amount can be used as cache memory. The more memory you commit, the better the performance will be.

1.3 License

This project is released under the Lesser General Public license, which means that you may use the library and it's sourcecode for any purpose you want, that you may link with it and use it commercially, but that ANY change to the code must be released under the same license. We would appreciate if you would send us a patch when you add support for new hardware, but this is not obligatory, since it falls under linking as far as the LGPL is concerned.

2 Getting started

2.1 On Linux (file)

Debugging efsl on embedded devices is a rather hard job, because you can't just printf debug strings or watch memory maps easily. Because of that, core development has been performed under the Linux operating system. Under Linux, efsl can be compiled as library and used as a userspace filesystem handler. On Unix- style operating system (like Linux), all devices (usb stick, disc, ...) can be seen as a file, and as such been opened by efsl.

In the following section, we will explain how to get started using efsl as userspace filesystem handler. However, please note that the main focus for efsl is to support embedded systems, which usually don't even have 1% of the memory you have on a PC. Accessing files on a FAT-filesystem with efsl will be much slower than when accessing these files with the Linux FAT kernel modules.

2.1.1 Download & Compile

Let's get started:

1. Get the latest release of efsl on http://www.sf.net/projects/efsl/ and put it in your homedir
2. Unpack the library (tar xvfj efsl-version.tar.bz2)
3. Get inside the directory (cd ~/efsl)
4. Create a symlink from Makefile-LINUX to Makefile (ln -s Makefile-LINUX Makefile)
5. Copy conf/config-sample-linux.h to conf/config.h (cp conf/config-sample-linux.h conf/config.h)
6. Compile the library (make lib)
7. Find the compiled filesystem library (libefsl.a) in the current directory

If you got any errors with the steps above, please check that that you have the following packages installed: tar, gcc, libgcc, binutils & make.

2.1.2 Example

Since efsl itself is only a library, it's not supposed to do anything out of the box, than just compile. To get started, we'll show here a small example program that opens a file on a disc/usb-stick/floppy that contains a FAT-filesystem and prints it's content to stdout.

First, create a new directory in which you put the compiled efsl-library (libefsl.a) and create a new file called linuxtest.c containing:

```
1  #include <stdio.h>
2  #include <efs.h>
3
4  int main(void)
5  {
6      EmbeddedFileSystem efs;
7      EmbeddedFile file;
8      unsigned short i,e;
9      char buf[512];
10
11     if(efs_init(&efs,"/dev/sda")!=0){
12         printf("Could not open filesystem.\n");
13         return(-1);
14     }
15
16     if(file_fopen(&file,&efs.myFs,"group",'r')!=0){
17         printf("Could not open file.\n");
18         return(-2);
19     }
20
21     while(e=file_read(&file,512,buf)){
22         for(i=0;i<e;i++)
23         printf("\%c",buf[i]);
24     }
25
26     return(0);
27 }
```

Some extra information on the code above:

- Line 1-2: The header files for stdio (used for printf) and efsl are included. When using the basic efsl functions, `efs.h` is the only header file of the efsl library that needs to be included.
- Line 6: The object efs is created, this object will contain information about the hardware layer, the partition table and the disc.
- Line 7: The object file is created, this object will contain information about the file that we will open on the efs-object.
- Line 9: A buffer of 512 bytes is allocated. This buffer will be filled by fread with data.
- Line 11-14: Call of `efs_init`, which will initialize the efs-object. To this function we pass:
    1. A pointer to the efs-object.
    2. A pointer to the file that contains the partition table / file system (in this example, we select a device as file).

If this function returns 0, it means that a valid fat partition is found on the device given. If no valid fat-filesystem is found, or the file does not exist, the function returns a negative value. In this example we then print an error message and quit.

- Line 16-19: Call of `file_fopen()` , which will initialize the file-object. To this function we pass:

  1. A pointer to the file-object.
  2. A pointer to the filesystem-object.
  3. A pointer to the filename.
  4. A char containing the the mode (read, write, append).

  If this function returns 0, it means the file has successfully been opened for reading / writing / appending. If the file could not be opened, a negative value is returned.

- Line 21-24: Call of `file_read()` , which will read a given value of bytes (in this example 512) from a file and put it's content into the buffer passed (in this example called buf). This function returns the amount of bytes read, so the while-loop will be executed as long as there are bytes left in the file. The code inside the while-loop will print all characters in the buffer.

2.1.3 Testing

So now let's test the program:

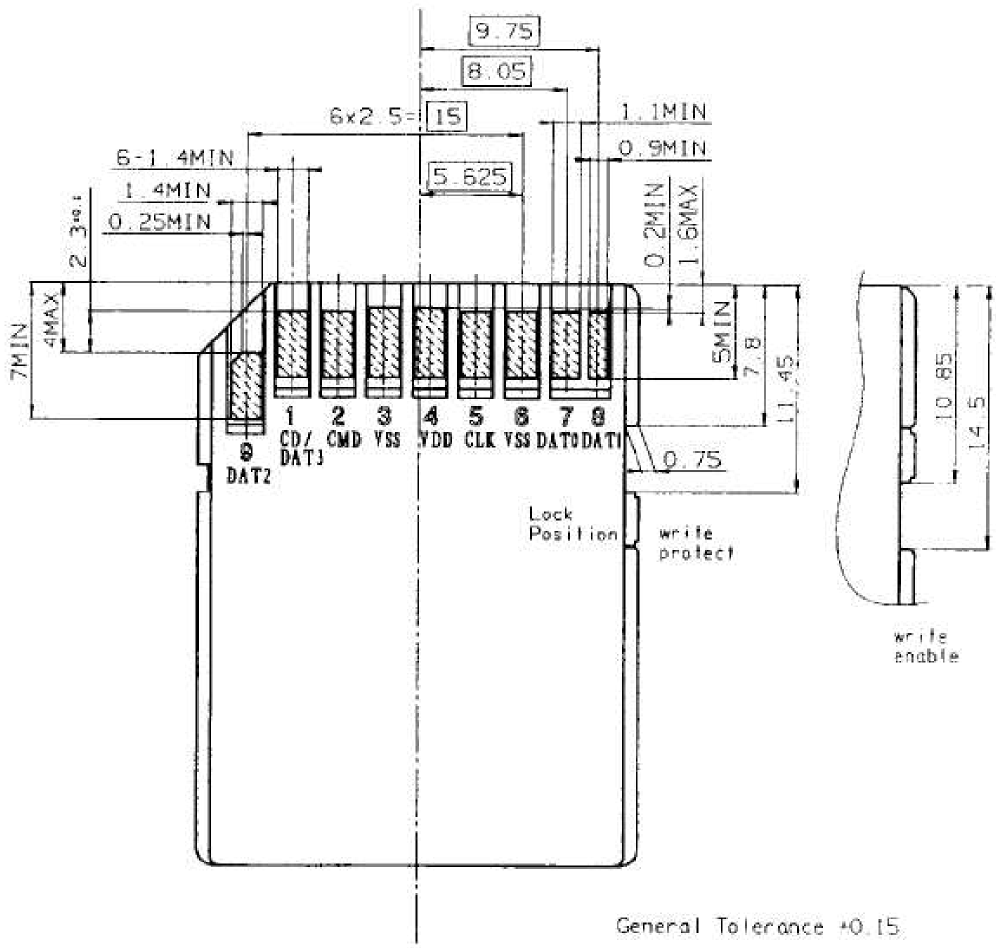
1. Compile the program (gcc -I/home/user/efsl/inc/ -I/home/user/efsl/conf -o linuxtest linuxtest.c -L./ -lefsl).
2. Insert a usb-disc, floppy, mp3-stick, ... with a valid fat-filesystem on it.
3. Mount the device, copy the file /etc/group on it's root dir & umount it.
4. Check that you have permission to access the device (chown username /dev/sda*)
5. Run the program (./linuxtest)

2.2 On AVR (SD-Card)

This section describes how to implement Efsl on a AVR μC connected to an SD-Card (SPI). For getting efsl to compile, the avr-gcc compiler and avr-libc library are required. On Windows you should install WinAVR (http://winavr.sourceforge.net/), on Linux you can install the packages separately (see http://www.nongnu.org/avr-libc/user-manual/install_tools.html for a nice howto).

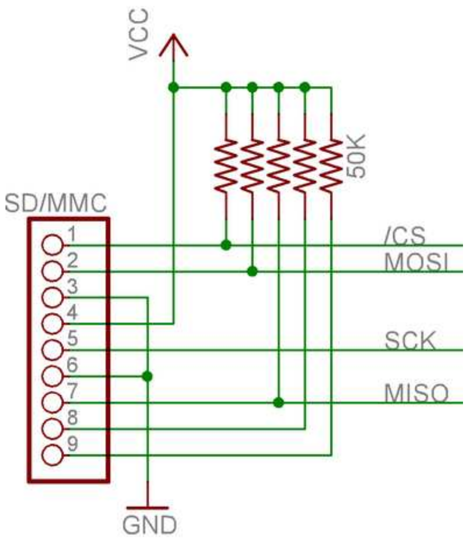
2.2.1 Hardware

First, you need set up a prototype in which you connect the CD, CMD, DAT0 & CLK lines from the SD-Card to /CS, MOSI, MISO & SCK from the Atmega.



Connect the following lines on the SD-card:

- Pin 9 (DAT2) - NC (or pull-up to 3.3V)
- Pin 1 (CD) - Any pin on the Atmega128
- Pin 2 (CMD) - MOSI (pin 12 on the Atmega128)
- Pin 3 (Vss) - GND
- Pin 4 (Vdd) - +3.3V
- Pin 5 (CLK) - SCK (pin 11 on the Atmega128)
- Pin 6 (Vss) - GND
- Pin 7 (DAT0) - MISO (pin 12 on the Atmega128)
- Pin 8 (DAT1) - NC (or pull-up to 3.3V)



Remark: this schematic includes pull-up's to 3.3V, which can be left off.

Remark 1: Make sure that your μC is running on 3,3V, so you don't damage your SD-Card.

Remark 2: CD is currently static set to PB0, but will become variable in future releases.

2.2.2 Download & Compile

Let's get started:

1. Get the latest release of efsl on http://www.sf.net/projects/efsl/
2. Unpack the library (on Windows, you can use WinACE or WinRAR)
3. Copy `Makefile-AVR` to `Makefile`
4. Copy `conf/config-sample-avr.h` to `conf/config.h`
5. Compile the library (`make lib`)

Now you should have `libefsl.a` in the efsl directory.

2.2.3 Example

Since Efsl itself is only a library, it's not supposed to do anything out of the box, than just compile. To get started, we'll show here a small example program that opens an existing file and writes the content to a new file.

First, create a new directory in which you put the compiled efsl-library (libefsl.a) and create a new file called avrtest.c containing:

```
1  #include <efs.h>
2
3  void hang(void);
4
5  void main(void)
6  {
7      EmbeddedFileSystem efs;
8      EmbeddedFile file_r, file_w;
9      unsigned short i,e;
10     char buf[512];
11
12     if(efs_init(&efs,0)!=0){
13         hang();
14     }
15
16     if(file_fopen(&file_r,&efs.myFs,"orig.txt",'r')!=0){
17         hang();
18     }
19
20     if(file_fopen(&file_w,&efs.myFs,"copy.txt",'w')!=0){
21         hang();
22     }
23
24     while(e=file_read(&file_r,512,buf)){
25         file_write(&file_w,e,buf);
26     }
27
28     file_fclose(&file_r);
29     file_fclose(&file_w);
30
31     fs_umount(&efs.myFs);
32
33     hang();
34 }
35
36 void hang(void)
37 {
38     while((1))
39         _NOP();
40 }
```

Some extra information on the code above:

- Line 1: The header file for efsl is included here. When using the basic efsl functions, efs.h is the only header file on the efsl library that needs to be included.

- Line 7: The object efs is created, this object will contain information about the hardware layer, the partition table and the disc.
- Line 8: The objects `file_r` and `file_w` are created, these objects will contain information about the files that we will open on the efs-object.
- Line 9: A buffer of 512 bytes is allocated. This buffer will be used for reading and writing blocks of data.
- Line 12: Call of `efs_init()` , which will initialize the efs-object. To this function we pass:

  1. A pointer to the efs-object.
  2. A pointer to the file that contains the partition table / file system (in this example, we select a device as file).

  If this function returns 0, it means that a valid fat partition is found on the SD-card connected. If no valid fat-filesystem is found, or the file does not exist, the function returns a negative value. In this example we then go to an infinite loop to prevent the program to continue.
- Line 16 & 20: Call of `file_fopen()` , which will initialize the file-objects. To this function we pass:

  1. A pointer to the file-object.
  2. A pointer to the filesystem-object.
  3. A pointer to the filename.
  4. A char containing the the mode (read, write, append).

  If this function returns 0, it means the file has successfully been opened for reading / writing / appending. If the file could not be opened (because for example a file already exists), a negative value is returned.
- Line 24: Call of `file_read()` , which will read a given value of bytes (in this example 512) from a file and put it's content into the buffer passed (in this example called buf). This function returns the amount of bytes read, so the while-loop will be executed as long as there are bytes left in the file.
- Line 25: Call of `file_write()` , which will write a given value of bytes (in this example, the amount of bytes that was read by `file_read()`) from the buffer passed to a file. This function returns the amount of bytes written.
- Line 28 & 29: Call of `file_fclose()` , which will close the file-objects.
- Line 31: Call of `fs_umount()` , which will write all buffers to the the SD-card.

2.2.4 Testing

So now let's test the program:

1. Make sure that your directory contains both the example from above called `avrtest.c` and the library `libefsl.a`.
2. Compile the program:
   - On Linux (with avr-gcc): avr-gcc -I/home/user/efsl/inc/ -I/home/user/efsl/conf -ffreestanding -mmcu=atmega128 -Os -o avrtest.o avrtest.c -L./ -lefsl
   - On Windows (with WinAVR): avr-gcc -Ic:\efsl\inc -Ic:\efsl\conf -ffreestanding -mmcu=atmega128 -Os -o avrtest.o avrtest.c -L.\ -lefsl
3. Generate a hexfile (avr-objcopy -j .text -j .data -O ihex avrtest.o avrtest.hex)
4. Connect an SD-card to your Atmega128 with a file called `orig.txt` on it.
5. Flash the hex file into your μC.
   - On Linux: avrdude -P /dev/ttyUSB0 -c stk500 -p m128 -Uflash:w:avrtest.hex
   - On Windows: use Atmel AVR-Studio
6. Reset your μC and wait some time (depending on how big the file `orig.txt` is).
7. Disconnect the SD-card, so you can put it in your card reader and find out if the file `orig.txt` is copied to `copy.txt`.

2.3 On DSP (SD-Card)

This section will tell you everything you need to know to start using the embedded filesystems library on a TMS Digital Signal Processor from Texas Instruments. The only thing that is required is that you have a McBSP port available, and that your DSP support CLOCKSTOP mode, which is required to connect a SPI compatible device.

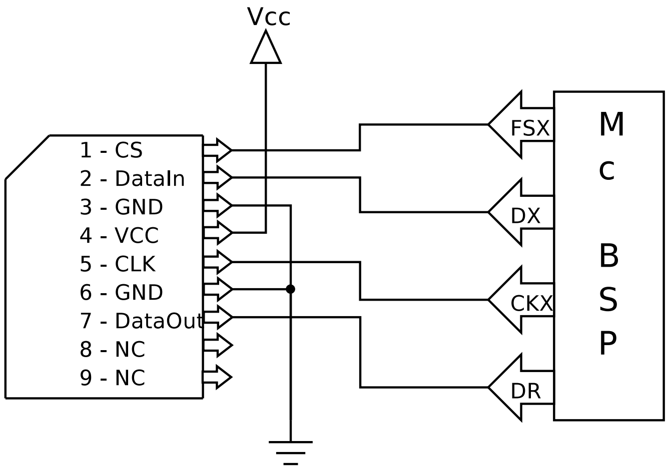
There are special DSP's from TI which have a special MMC/SD card controller, if you want to use this special interface you will have to create a hardware endpoint for it. This section only describes connecting an SD card to a normal McBSP port, since every TI DSP has at least one of them.

2.3.1 Hardware

Connecting the SD card to the McBSP is straightforward, you will have to make 4 data related connections, Vcc and ground, resulting in a 6 wire interface.

SD Card Interface			McBSP Interface	
1	CS	Chip select	FSX	Frame Sync Transmit
2	MOSI	Master out Slave In	DX	Data transmit
3	GND	Supply Ground		
4	Vcc	Supply voltage (3.3 Volt)		
5	Clk	Clock	CLKX	Clock Transmit
6	GND	Supply ground		
7	MISO	Master in Slave out	DR	Data receive
8	NC	Not connected		
9	NC	Not connected		

You can optionally pull the DataIn and DataOut lines up to Vcc with a $10k\Omega$ resistor, but we found that this was not required for operation.



The frame sync from the McBSP port is used to select the card whenever a databyte has to be transferred, it is connected to the chip select of the SD card. The DX and DR pins are connected to the SDcard's DataIn and DataOut lines respectively. Finally the McBSP will have to generate a clock for the SDcard so that it can perform operations, this is accomplished by connecting the clock transmit line of the McBSP port to the CLK pin of the SDCard.

2.3.2 McBSP configuration

McBSP Register Explanations			
SPCR			Serial Port Control Register
Name	Bit	Value	Value `(0x00001800 | 0x00410001)`
RRST	`0`	`1b`	The serial port receiver is enabled
XRST	`16`	`1b`	The serial port transmitter is enabled
CLKSTP	`12:11`	`11b`	Clock starts on falling edge without delay(see CLKXM)
GRST	`22`	`1b`	Sample rate generator is pulled out of reset
PCR			Pin Control Register
Name	Bit	Value	Value `0x00000A0C`
CLKXP	`1`	`0b`	Transmit data on the rising edge ofthe clock
FSXP	`3`	`1b`	Frame Sync (Chip select on SD card) is active low
CLKXM	`9`	`1b`	McBSP is a master in SPI mode and generates the clock based on the sample rate generator
FSXM	`10`	`1b`	Frame sync is determined by tge sample rate generator
RCR/XCR			Receive/Transmit Control Register
Name	Bit	Value	Value `0x00010000`
RWDLEN	`7:5`	`000b`	Receive element is 8 bits (1byte) large
XDATDLY	`17:16`	`01b`	1 bit data delay (after frame sync)
SRGR			Sample Rate Genrator
Name	Bit	Value	Value `0x20000002`
CLKSM	`29`	`1b`	The sample rate generator clock is derived from the internal clock
FSGM	`28`	`0b`	The transmit frame sync signal is generated on every DXR to XSR copy
CLKGDV	`7:0`	`0x02h`	The clock divider

3 Configuring EFSL

In this section we're going to talk about the configuration file (config.h), that defines the behavior of the library. In the configuration files there are many settings, most of which default to safe or 'standard' compliant settings.

For every platform we try to deliver a sample configuration, with setting tweaked for that architecture. This documentation only refers to the general elements which are tied to the library rather that the target hardware.

3.1 Hardware target

Here you will define what kind of hardware you will be using. Please refer to section 5.3 to learn how to write a hardware endpoint. Here you must `#define` the name of your hardware endpoint. The following list contains the endpoints that the library ships with.

`HW_ENDPOINT_LINUX`	This endpoint uses a regular file as a "disc" containing a filesystem. This is a great endpoint for testing and debugging. All development is done using this emulation.
`HW_ENDPOINT_ATMEGA128_SD`	This endpoint is for the Atmel ATMega 128 with an SD card attached to the SPI pins of the device. Several settings that are specific for this endpoint can be found in the AVR sample configuration. A Makefile is also provided for compiling the EFSL library using avr-gcc.
`HW_ENDPOINT_DSP_TI6713_SD`	This endpoint is for a TI DSP, it should work with any McBSP port, due to the infinite amount of options, you should refer to the source code of this endpoint for fine tuning, or selecting what port to use (defaults to McBSP0).

3.2 Memory configuration

This section only has one option, called `BYTE_ALIGNMENT` . If you define this keyword the library will assume that your CPU is capable of accessing the memory in any way it sees fit. This is the case on AVR, because they are 8 bit processors, and it is also the case on Intel x86 hardware. Both architectures can read and write words, or double words on any location in memory, be it word aligned or not.

However, some CPU's, are not capable of doing this, and require that all double words are aligned on a double word boundary, and all word are aligned on a word boundary. This causes problems with some of the casts that are performed in EFSL. If you have such a CPU, then you must comment this option out. The effect is that special functions will be used to copy or cast memory. These functions work around the problem by using memCpy, or manually copying elements of the structs that are normally cast when `BYTE_ALIGNMENT` is defined.

If you have an 8 bit architecture, or are running on PC, there is no need to turn this off. If you do, the library will work fine, and maybe even without slowdown.

On architectures that do have the alignment problem, you should turn this flag off. Failure to do so will result in undefined behavior.

3.3 Cache configuration

This section is dedicated to configuring the cache memory for the library. Caching is performed by the IOMan object, see section 5.4.

IOMAN_NUMBUFFER

This number determines how much memory will be used for caching. Since this is sector based one `IOMAN_NUMBUFFER` equals to 512 byes of memory, plus a small overhead in settings (approximately 8 bytes). This number is also affected by `IOMAN_NUMITERATIONS`.

You should carefully consider how much memory you will dedicate to caching. A too low number will cause excessive data transfer to and from the disc, where a too high number will simply be a waste of memory.

A good rule of thumb is to use 1 buffer per filesystem you create, and 2 buffers per file you want to use simultaneously. So for a simple application with one filesystem, and one file operation, 2 or 3 buffers will be fine. If you have memory to spare, you can use 6 buffers. Using more buffers will have a minimal effect on performance.

If you want to seek and rewrite portions of a file, add an extra buffer for that file. Using the list function or creating directories will be disc intensive, try to smoothen it by using an extra 3 buffer for either operation.

It is perfectly possible to have multiple files op for reading and writing, on different filesystems, with listing etc and only using 1 buffer. It will be a tough blow on performance though.

IOMAN_NUMITERATION

This number controls how many stack places each cache place gets. Refer to the IOMan section for an explanation. In short, if you only have 1 buffer, leave it at 3. If you use more than 4 buffers try decreasing the number to 2 or 1 for a small memory gain.

If you get errors, it means you have set it too low (see error support). It is best to leave this at the default setting (do not increase it), unless you know what you are doing.

IOMAN_DOMEMALLOC

This configures how IOMan will get it's memory. If you leave it enable, the memory will be allocated by IOMan itself. That means that when you declare the IOMan object it will have a member the size of 512·IOMAN_NUMBUFFER. That also means that that huge lump of memory will reside on the stack. On a true embedded platform with no malloc, this is your best option. The last argument of `ioman_init` will be ignored.

If you comment this out,IOMan will take a `euint8*` pointer as it's third argument to `ioman_init`. It will use the memory pointed to as cache. You will have to make sure it's reserved and of the correct size. This allows you to put the memory on the heap, or perform special tricks like deallocating it without having to umount your filesystem and open files. On systems with malloc, this is the recommended setting.

If you use the efs wrapper object, please look at the `efs_init` documentation on how to pass the ioman pointer.

3.4 Pre-allocation

Our VFAT module supports the concept of pre-allocation. When writing files, for example log files, it is usually done with tiny bits a time. That is not the most efficient way, but it is usually the only solution that works on embedded systems. Every time you cross a cluster boundary with your write, the library has to search a new cluster (reading the FAT), allocate it (write to the FAT).

Clearly, this is a waste. The solution we came up with was preallocating. This means that when you write to a file, and fwrite sees that it needs to allocate more clusters, it will allocate too many of them. Since this is done in one operation, it requires usually only one read and one write to the FAT. This can save up to 50% disc I/O in some applications.

The drawback is that the allocation happens in larger chunks, if you do this with many files, you might end up with larger than normal amounts of slackspace.

We have also implemented this feature for directories. This is very useful if you have to create a lot of small files, since the directories grow by larger portions then.

CLUSTER_PREALLOC_FILE

This number determines the default value of extra clusters that will be allocated with every sizeincrease. For example, if fwrite calculates that it needs 7 clusters, and `CLUSTER_PREALLOC_FILE` is 30 then efsl will allocate 37 clusters. This means (assuming every write needs 7 clusters) that the next 4 writes won't require any write operation to the FAT (and due to the cluster cache the FAT will probably have to be read only once).

The value you put here will be the default value, it can be changed per file object. (not yet implemented).

CLUSTER_PREALLOC_DIRECTORY

The same explanation as above counts, only this value is used for directories. Generally you should not put this above 10 (unless your speed tests prove otherwise off course).

3.5 Endianness

The Microsoft FAT filesystem was originally created to be run on Intel compatible hardware. Therefore the Microsoft programmers decided to record all data on the disc in little endian format. Our library supports running on big endian devices. Here you can select whether your target CPU is little or big endian.

Running on big endian will cause some performance lose because (rather simple) calculations have to be made to all numbers that have to interpreted by the library. This does not apply to data within the files off course.

If the flag `#LITTLE_ENDIAN` is set, efsl will assume that your hardware is little endian. If you have a big endian system, you should comment this out. The function `fs_checkEndian` will tell you if you have selected the right endianness, this is a check you might want to use.

3.6 Date and time

This flag determines if you want to have date and time support. With date and time support we mean that when you create or update a file the directory entry will receive the correct date and time stamp.

Please refer to section 4.1 to learn more about how this works.

If you disable date and time support by commenting the `#DATE_TIME_SUPPORT` then all dates and times that need to be created or updated will be set to zero, which in FAT land corresponds to the first of January of the year 1970.

3.7 Errors

When the library encounters an error, there be an error cascade moving from the error-causing object to the topmost object where the request started. Seen from userland this gives you extremely little information, usually nothing more than fail or success.

Every object in the library has an optional error field, that contains a unique number that corresponds to a specific error. If you examine every error field you can see exactly where the error was started and what the effect was on the higher level objects.

In a more practical sense you can display an error number or explanation to your users, giving yourself or them a better chance to correct or avoid the problem. Please see the section on error on what every value means.

3.8 Debug

This will turn debug support on or off. When enable (and your platform has a means of output that is supported by EFSL) it you will see messages you have created yourself, or that are printed by the library. By default the library is very silent, only very critical errors might get printed out.

This option is depreciated and is left in for backward compatibility.

4 EFSL Functions

4.1 Date and time support

The EFSL library supports setting and updating all date and time fields supported by the filesystem. In order to do this the library must know the current time and date at all times. Since it has to run everywhere, there is no standard mechanism to get the date/time, and some systems do not have a clock.

With default configuration there is no date or time support, you have to turn it on manually in the configuration file `config.h` . You will have to uncomment the field named `#define DATE_TIME_SUPPORT` , in order to activate date/time support.

Furthermore you will have to provide the library with date and time information. A set of defines was used for this, when date/time support is not enabled, the defines automatically return `0x0000` for all time and date fields, so there is no performance suffer when you do not need date/time support. If you do need it you will have to provide 6 functions to the library that will tell it the time. Since these functions may get called often, it is highly recommended that you cache the time result somewhere so you can serve the library directly from ram. If you do not do this and your RTC request take a lot of time, you may suffer large losses in read or write operations depending on your hardware.

The six functions are:

- `euint16 efsl_getYear(void)`
- `euint8 efsl_getMonth(void)`
- `euint8 efsl_getDay(void)`
- `euint8 efsl_getHour(void)`
- `euint8 efsl_getMinute(void)`
- `euint8 efsl_getSecond(void)`

Internally the library will recalculate these numbers to match the filesystem that is currently in use.

4.2 efs_init

Purpose

Initializes the hardware and the software layer.

Prototype

`esint8 efs_init(EmbeddedFileSystem *efs, eint8* opts);`

Arguments

Objects passed to `efs_init` :

- `efs` : empty EmbeddedFileSystem object
- `opts` : character string containing options, depending on what interface you are using:
  - Linux: opts points to the path to the device
  - AVR: opts points to the card enable pin (TODO)
  - DSP: opts points to the card enable memory address (TODO)

Return value

Returns 0 if no errors are detected.

Returns non-zero if a low-level error is detected:

- Returns -1 if the interface could not be initialized.
- Returns -2 if the filesystem could not be initialized.

Example

```
#include "efs.h"

void main(void)
{
    EmbeddedFileSystem efsl;
    esint8 ret;

    DBG((TXT("Will init efsl now.\n")));
    ret=efs_init(&efsl,"/dev/sda");
    if(ret==0)
        DBG((TXT("Filesystem correctly initialized.\n")));
    else
        DBG((TXT("Could not initialize filesystem (err \%d).\n"),ret));
}
```

4.3 file_fopen

Purpose

Searches for file and initializes the file object.

Prototype

`esint8 file_fopen(File *file, FileSystem *fs, eint8 *filename, eint8 mode);`

Arguments

Objects passed to `file_fopen` :

- `file` : pointer to a File object
- `fs` : pointer to the FileSystem object
- `filename` : pointer to the path + filename
- `mode` : mode of opening, this can be:
  - 'r': open file for reading
  - 'w': open file for writing
  - 'a': open file for appending

Return value

Returns 0 if no errors are detected.

Returns non-zero if an error is detected:

- Returns -1 if the file you are trying to open for reading could not be found.
- Returns -2 if the file you are trying to open for writing already exists.
- Returns -3 if no free spot could be found for writing or appending.
- Returns -4 if mode is not correct (if it is not 'r', 'w' or 'a').

Example

```
#include "efs.h"

void main(void)
{
    EmbeddedFileSystem efsl;
    File file_read, file_write;

```

```
    /* Initialize efsl */
    DBG((TXT("Will init efsl now.\n")));
    if(efs_init(&efsl,"/dev/sda")!=0){
        DBG((TXT("Could not initialize filesystem (err \%d).\n"),ret));
        exit(-1);
    }
    DBG((TXT("Filesystem correctly initialized.\n")));

    /* Open file for reading */
    if(file_fopen(&file_read, &efsl.myFs, "read.txt", 'r')!=0){
        DBG((TXT("Could not open file for reading.\n")));
        exit(-1);
    }
    DBG((TXT("File opened for reading.\n")));

    /* Open file for writing */
    if(file_fopen(&file_write, &efsl.myFs, "write.txt", 'w')!=0){
        DBG((TXT("Could not open file for writing.\n")));
        exit(-2);
    }
    DBG((TXT("File opened for writing.\n")));

    /* Close files & filesystem */
    fclose(&file_read);
    fclose(&file_write);
    fs_umount(&efs.myFs);
}
```

4.4 file_fclose

Purpose

Updates file records and closes file object.

Prototype

`esint8 file_fclose(File *file);`

Arguments

Objects passed to `file_fopen` :

- `file` : pointer to a File object

Return value

Returns 0 if no errors are detected.

Returns non-zero if an error is detected.

Example

```
#include "efs.h"

void main(void)
{
    EmbeddedFileSystem efsl;
    File file;

    /* Initialize efsl */
    DBG((TXT("Will init efsl now.\n")));
    if(efs_init(&efsl,"/dev/sda")!=0){
        DBG((TXT("Could not initialize filesystem (err %d).\n"),ret));
        exit(-1);
    }
    DBG((TXT("Filesystem correctly initialized.\n")));

    /* Open file for reading */
    if(file_fopen(&file, &efsl.myFs, "read.txt", 'r')!=0){
        DBG((TXT("Could not open file for reading.\n")));
        exit(-1);
    }
    DBG((TXT("File opened for reading.\n")));

    /* Close file & filesystem */
    fclose(&file);
```

```
        fs_umount(&efs.myFs);
    }
```

4.5 file_read

Purpose

Reads a file and puts it's content in a buffer.

Prototype

`euint32 file_read (File *file, euint32 size, euint8 *buf);`

Arguments

Objects passed to `file_read` :

- `file` : pointer to a File object
- `size` : amount of bytes you want to read / put in buf
- `buf` : pointer to the buffer you want to store the data

Return value

Returns the amount of bytes read.

Example

```
#include "efs.h"

void main(void)
{
    EmbeddedFileSystem efsl;
    euint8 buffer[512];
    euint16 e, f;
    File file;

    /* Initialize efsl */
    DBG((TXT("Will init efsl now.\n")));
    if(efs_init(&efsl,"/dev/sda")!=0){
        DBG((TXT("Could not initialize filesystem (err \%d).\n"),ret));
        exit(-1);
    }
    DBG((TXT("Filesystem correctly initialized.\n")));

    /* Open file for reading */
    if(file_fopen(&file, &efsl.myFs, "read.txt", 'r')!=0){
        DBG((TXT("Could not open file for reading.\n")));
        exit(-1);
    }
    DBG((TXT("File opened for reading.\n")));
```

```
	/* Read file and print content */
	while((e=file_read(&file,512,buffer))){
		for(f=0;f<e;f++)
			DBG((TXT("\%c"),buffer[f]));
	}

	/* Close file & filesystem */
	fclose(&file);
	fs_umount(&efs.myFs);
}
```

4.6 file_write

Purpose

Reads a file and puts it's content in a buffer.

Prototype

`euint32 file_write(File *file, euint32 size, euint8 *buf)`

Arguments

Objects passed to `file_read` :

- `file` : pointer to a File object
- `size` : amount of bytes you want to write
- `buf` : pointer to the buffer you want to write the data from

Return value

Returns the amount of bytes written.

Example

```
#include <string.h>
#include "efs.h"

void main(void)
{
    EmbeddedFileSystem efsl;
    euint8 *buffer = "This is a test.\n";
    euint16 e=0;
    File file;

    /* Initialize efsl */
    DBG((TXT("Will init efsl now.\n")));
    if(efs_init(&efsl,"/dev/sda")!=0){
        DBG((TXT("Could not initialize filesystem (err %d).\n"),ret));
        exit(-1);
    }
    DBG((TXT("Filesystem correctly initialized.\n")));

    /* Open file for writing */
    if(file_fopen(&file, &efsl.myFs, "write.txt", 'w')!=0){
        DBG((TXT("Could not open file for writing.\n")));
        exit(-1);
    }
```

```
        DBG((TXT("File opened for reading.\n")));

        /* Write buffer to file */
        if( file_write(&file,strlen(buffer),buffer) == strlen(buffer) )
            DBG((TXT("File written.\n")));
        else
            DBG((TXT("Could not write file.\n")));

        /* Close file & filesystem */
        fclose(&file);
        fs_umount(&efs.myFs);
}
```

4.7 mkdir

Purpose

Creates a new directory.

Prototype

`esint8 mkdir(FileSystem *fs,eint8* dirname);`

Arguments

Objects passed to `mkdir` :

- `fs` : pointer to the FileSystem object
- `dir` : pointer to the path + name of the new directory

Return value

Returns 0 if no errors are detected.

Returns non-zero if an error is detected:

- Returns -1 if the directory already exists.
- Returns -2 if the path is incorrect (parent directory does not exists).
- Returns -3 if no free space is available to create the directory.

Example

```
#include "efs.h"

void main(void)
{
    EmbeddedFileSystem efsl;

    /* Initialize efsl */
    DBG((TXT("Will init efsl now.\n")));
    if(efs_init(&efsl,"/dev/sda")!=0){
        DBG((TXT("Could not initialize filesystem (err %d).\n"),ret));
        exit(-1);
    }
    DBG((TXT("Filesystem correctly initialized.\n")));

    /* Create new directories */
    if(mkdir(&efs.myFs,"dir1")==0){
        mkdir(&efs.myFs,"dir1/subdir1");
```

```
			mkdir(&efs.myFs,"dir1/subdir2");
			mkdir(&efs.myFs,"dir1/subdir3");
		}

		/* Close filesystem */
		fs_umount(&efs.myFs);
	}
```

4.8 ls_openDir

Purpose

This function opens a directory for viewing, allowing you to iterate through it's contents.

Prototype

```
esint8 ls_openDir(DirList *dlist,FileSystem *fs,eint8* dirname);
```

Arguments

Objects passed to `ls_openDir` :

- `dlist` : pointer to a DirList object
- `fs` : pointer to the FileSystem object
- `dirname` : C string containing the directorypath

Return value

This function will return 0 when it has opened the directory, and -1 on error.

Example

```c
#include "efs.h"
#include "ls.h"

	void main(void)
	{
		EmbeddedFileSystem efsl;
		DirList list;

		/* Initialize efsl */
		if(efs_init(&efsl,"/dev/sda")!=0){
			DBG((TXT("Could not initialize filesystem (err %d).\n"),ret));
			exit(-1);
		}

		/* Open the directory */
		ls_openDir(list,&(efsl.myFs),"/usr/bin/");

		/* Correctly close the filesystem */
		fs_umount(&efs.myFs);
	}
```

Please note that it is not required to close this object, if you wish to switch to another directory you can just call `ls_openDir` on the object again.

4.9 ls_getNext

Purpose

This function fetches the next valid file in the current directory and copies all relevant information to `dirlist->currentEntry` .

Prototype

```
esint8 ls_getNext(DirList *dlist);
```

Arguments

Objects passed to `ls_getNext` :

- `dlist` : pointer to a DirList object

Return value

This function will return 0 when it has found a next file in the directory, and was successful in copying it to `dirlist->currentEntry` . It will return -1 when there are no more files in the directory.

Example

To browse through a directory you should first open it with `ls_openDir` and then you can call `ls_getNext` in a loop to iterate through the files. Please note that they are unsorted.

```
#include "efs.h"
#include "ls.h"

void main(void)
{
        EmbeddedFileSystem efsl;
        DirList list;

        /* Initialize efsl */
        if(efs_init(&efsl,"/dev/sda")!=0){
            DBG((TXT("Could not initialize filesystem (err \%d).\n"),ret));
            exit(-1);
}

        /* Open the directory */
        ls_openDir(list,&(efsl.myFs),"/usr/bin/");

        /* Print a list of all files and their filesize */
        while(ls_getNext(list)==0){
            DBG((TXT("%s (%li bytes)\n"),
```

```
                list->currentEntry.FileName,
                list->currentEntry.FileSize));
        }

        /* Correctly close the filesystem */
        fs_umount(&efs.myFs);
}
```

Please note that it is not required to close this object, if you wish to switch to another directory you can just call `ls_openDir` on the object again.

5 Developer notes

5.1 Integer types

Standard C data types have the annoying tendency to have different sizes on difference compilers and platforms. Therefore we have created 9 new types that are used everywhere throughout the library. When you implement your platform you should check if any of the existing one matches your hardware, or create a new one.

Here's an overview:

Type	**Size**	**Signedness**
`eint8`	1 byte	default to platform
`esint8`	1 byte	signed
`euint8`	1 byte	unsigned
`eint16`	2 bytes	default to platform
`esint16`	2 bytes	signed
`euint16`	2 bytes	unsigned
`eint32`	4 bytes	default to platform
`esint32`	4 bytes	signed
`euint32`	4 bytes	unsigned

You will find the relevant code in the file `types.h` in the directory `inc/` .

5.2 Debugging

Since debugging on every device is completely different, a DBG macro is implemented. On Linux for example, this macro will print the string given to the screen (using printf). On AVR, it will send debug strings through the UART. For compatibility with other devices, it is necessary that you always use the DBG-macro instead of a device-specific debugging commands.

Because AVR-GCC puts strings in sram memory by default, every string should be surrounded by the TXT-macro. On AVR, this macro will put the string in program memory (flash), on any other device, this macro will be ignored.

Example of a debug string:
`DBG((TXT("This is test nr %d of %d.\n"),id,total));`

5.2.1 Debugging on Linux

On linux, debugging strings are sent to stdout using printf.

To enable debugging, set DEBUG in `config.h` .

5.2.2 Debugging on AVR

On AVR, debugging strings are sent through the UART and can be read using a terminal like minicom (linux) or hyperterminal (windows). Standard, the first UART is used, but this can be changed in `debug.c` to the second UART.

To enable debugging:

- Set DEBUG in `config.h`
- Set CLK to the clock speed of your AVR in `config.h`
- Set BAUDRATE to the baudrate you want in `config.h`
- Initialize debugging in your program by calling `debug_init()`

Remark: when you use the serial port in your main program, make sure you use a different UART than the one efsl is using when sending debug string.

5.2.3 Debugging on DSP

On DSP, debugging strings are sent to Code Composer using the printf function.

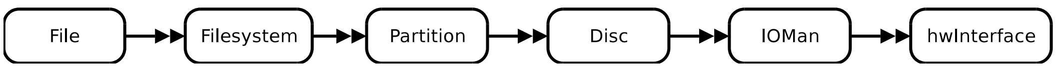
To enable debugging, set DEBUG in `config.h` .

Remark: this will only work when using a DSK-kit.

5.3 Adding support for a new endpoint

This section will describe step by step how to write an hardware endpoint. You will be required to write your own endpoint in case non of the existing endpoints matches your hardware.

First let's have a look at how EFSL is structured internally.



As you can see we have created a linear object model that is quite simple. The file en filesystem object deal with handling the filesystem specific stuff. Below that we find the Partition object that is responsible for translating partition relative addressing into disc-based LBA addressing.

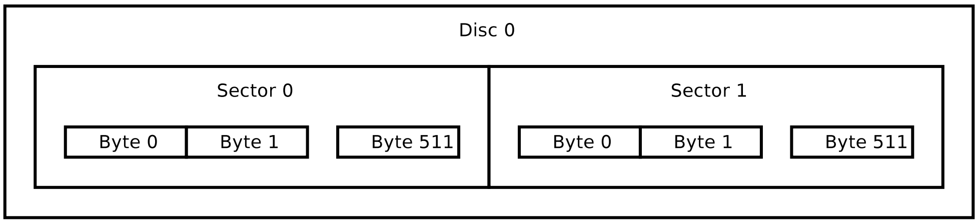
The Disc object hold the partition table, and has a direct link to a cache manager, IOMan. In IOMan, all requests for disc sectors come together. IOMan will perform checks to see if sectors have to be read from disc (or from memory), or written back to disc. In the latter case (reading or writing to disc), a request is made to the hardware layer.

The hardware interface has 3 responsibilities :

- Initialize the hardware

- Read sectors from disc
- Write sectors to disc

All requests are *sector*based, a sector is a 512 byte piece from the disc, that is aligned to a 512 byte boundary.



In this example we will create a new endpoint that will add support for data over pigeon carrier for the EFSL. Initializing the hardware will require feeding the pigeon and telling it where the data is. Reading/Writing will entail giving the bird the sector and letting it fly.

Perform the following steps:

1. Choose a name for your endpoint
   You will need this name to create the required defines in the source code. For our example I've chosen the name `PIGEON_CARRIER` . For consistency the final name is then `HW_ENDPOINT_PIGEON_CARRIER` .

2. Verify the sizes of integers
   Open inc/types.h and create a new entry for pigeon carriers. Perhaps one of the existing sets is identical to yours and you can copy-paste it.

3. Add your endpoint to interface.h
   Locate the file interface.h located in the directory inc/ Add a pigeon entry (located above the `#else ... NO INTERFACE DEFINED`)

```
#if defined(HW_ENDPOINT_0)
        #include "interfaces/0.h"
#elif defined(HW_ENDPOINT_1)
        #include "interfaces/1.h"
#elif defined(HW_ENDPOINT_PIGEON_CARRIER)
        #include "interfaces/pigeon.h"
#else
        #error "NO INTERFACE DEFINED - see interface.h"
#endif
```

4. Select your endpoint in conf/config.h

5. Create your sourcefiles
   Create a header file in inc/ and a sourcefile in src/interfaces . In this example I'm using pigeon.h and pigeon.c .

6. Add your object file to the Makefile Take the Makefile that works best on your platform (they should all work with GNU/Make), or create a new

one, using the existing one's as a template. Make sure to include your new pigeon object to the library. If you have an 'ar' like utility you can create a static library, else you may have to create a new project containing all required source files.

The basic framework is now complete, now all that's left to do is to write the code that will perform the actual flying work.

5.3.1 hwInterface

This structure represents the underlying hardware. There are some field that are required to be present (because EFSL uses them), but you may put in as much or a little as your driver requires to access the hardware.

As always in embedded design it is recommended to keep this structure as small as possible.

Example:

```
struct hwInterface{
    /* Field created for THIS hardware */
    Pigeon pigeon;

    /* Obligatory fields */
    euint32 sectorCount;
};
typedef struct hwInterface hwInterface;
```

5.3.2 if_initInterface

This function will be called one time, when the hardware object is initialized by `efs_init()` . This code should bring the hardware in a ready to use state.

The function's prototype is
`esint16 if_initInterface(hwInterface *hw, euint8* opts);`

Optionally but recommended you should fill in the hw-¿sectorCount field with the number of sectors. This field is used to validate sectorrequests.

An example of a initInterface function :

```
esint16 if_initInterface(hwInterface *hw, euint8* opts)
{
    /* Parse options */
    parse_options(opts); /* Your application may not need options */

    /* Check hardware state */
    if(!alive(hw->pigeon)){
        //printf("Pigeon died! :-(\n");
        return(DEAD_PIGEON); /* #define DEAD_PIGEON -1 */
    }

    /* Initialize hardware */
```

```
    feed(hw->pigeon);
    pet (hw->pigeon);

    /* Get sectors count */
    hw->numSectors = ask_pigeon_num_sectors(hw->pigeon);

    return(0);
}
```

5.3.3 if_readBuf

This function is responsible to read a sector from the disc and store it in a user supplied buffer. You will receive the hardware object, an address and a pointer to memory for storing the buffer.

Please be very careful to respect the boundaries of the buffers, since it will usually be IOMan calling this function, and if you have a buffer overflow you might corrupt the cache of the the next buffer, which in turn may produce extremely rare and impossible to retrace behavior.

The function prototype is:
`esint16 if_readBuf(hwInterface *hw,euint32 address, euint8* buf);`

The address is an LBA address, relative to the beginning of the disc. Should you be accessing an old hard disc, or a device which uses some other form of addressing you will have to recalculate the address to your own addressing scheme. Please note that there is no support for sectors that are not 512 bytes large.

```
esint8 if_readBuf(hwInterface* hw,euint32 address,euint8* buf)
{
    Message new_message;

    new_message.address = address;
    new_message.command = READ;

    pigeon_send(hw->pigeon,new_message); /* Launches the pigeon */
    while(!pigeon_returned(hw->pigeon)); /* Wait until the bird is back */
    memcpy(new_message.data,buf,512); /* Copy buffer */
    return(0);
}
```

5.3.4 if_writeBuf

The function `if_writeBuf` works exactly the same as it's reading variant.

5.4 I/O Manager

The IOManager that is the second lowest layer of the embedded filesystems library is responsible for coordinating disk input and output, as well as managing a caching system. This documentation describes the second implementation of IOMan, which includes features such as :

- Delayed write
- Buffer reference statistics
- Buffer exportable to users
- Support for cached direct I/O as well as indirect I/O
- Can allocate memory itself (on the stack), or you can do it yourself (heap)

5.4.1 General operation

Because of the limited memory nature of most embedded devices for which this library is intended several design decisions were made to minimize memory usage. Some of these required that some concessions be made. One of them is that there is no memory protection, since most devices don't have the memory to support this, or lack the ability to protect memory.

When IOMan receives a request for a sector, it will make sure it has the sector in it's own memory cache and then give the caller a `euint8*` pointer to that cache. The user is then free to do operations on that memory, and when it is done it should tell IOMan so. Several things can go wrong with this: you can request a sector for reading, and then write in the cache, thereby corrupting it. Or you can request a sector, but never release it (sort of a memory leak), which may result in very bad performance, and a deadlocked I/O manager.

But, taking into account that very little memory is required for operation, if you follow the I/O man rules, you will get a pretty clever caching object that will make writing new filesystems a simple job.

5.4.2 Cache decisions

Whenever ioman receives a request to fetch a sector, be it read or write, it will have to make sure it has, or can get the sector you want. It follows a certain path to do this.

1. First of all it will scan it's cache range to see if it already has the sector. If it is found, and it was a write request, the cache is marked writable. Usage and reference get incremented and a pointer is then returned to the requester. If the buffer cannot be found, ioman proceeds to step 2.
2. When an item is not in cache, it has to be fetched from the disc, the best place to store it is in memory that does not contain anything useful yet. Ioman will search for a place that is currently not occupied by anything. If it is found, the sector will be placed on that spot and a pointer returned. Else, ioman proceeds to step 3.

3. Since there is no other choice than to overwrite an already existing cache, ioman will try to find one that is the least interesting. First it will search for caches that are marked not writable, and have no users. Ioman will then select the one that has the least references. If there are none, it will search for caches that don't have users and are writable. Once again the one with the least references is returned. Since it is writable ioman will flush it to disc first. After that the requested sector is put there and a pointer returned. If it cannot find any caches that have no users it will go to step 4.

4. Since every cache spot is in use ioman will have to select one for overallocation. Since this selection depends on many factors and is rather complex, a points system is used. The algorithm considers every cache place and allocated a certain number of points to it, lower means that it is a better candidate for overallocation. Fifty percent of the points goes to the cache being marked writable, since having to write a sector is expensive. Another 35 percent goes to how many overallocations have already been done on that spot. It doesn't make sense to always overalloc the same buffer, it is better to spread this. The remaining 15 percent is determined by the number of references to the sector.

   After a function has selected the best candidate, ioman will overwrite that spot with the new sector. It will also push the status and sectornumber onto that cache's retrieval stack, so that when the sector is released, the older sector can be retrieved. If this fails go to step 5.

5. When ioman gets here it will return a (nil) pointer and flag an error.

5.4.3 Functions

I/O Manager Functions	
`ioman_init`	`esint8 (IOManager *ioman, hwInterface *iface, euint8* bufferarea)`
This function is called to initialize the internal state of the I/O manager. It should be the first function you call on an ioman object. Failure to do so will result in undefined behavior. The function clears all internal variables to a default safe state, and sets up it's memory region. There are two possibilities, if you supply a 0 pointer then a function will be called that contains a static variable with a size of 512 * `IOMAN_NUMBUFFERS` , else, it will be assumed that you allocated that memory yourself and the pointer you provided will be used.	
`ioman_reset`	`void (IOManager *ioman)`
This function is called from the initialization function, it does the actual reset of all variables.	
`ioman_pop`	`esint8 (IOManager *ioman,euint16 bufplace)`
This function fetches settings (sector number, usage and status register) from stack `bufplace` and puts it back on the main registers. It will return 0 on successful pop, and -1 on error, or when there are no elements to pop.	
`ioman_push`	`esint8 (IOManager *ioman,euint16 bufplace)`

I/O Manager Functions (continued)	
This function pushes the settings of cache `bufplace` onto that cache's stack. It does not destroy the data in the main registers. It will return 0 for a successful push, and -1 on error, or when there is no more space to push a new element.	
`ioman_readSector`	`esint8 (IOManager *ioman,euint32 address,euint8* buf)`
This function does the actual reading from the hardware, it is the one and only function that calls `if_readBuf()` , here a retry on failure policy could be implemented. This function will correctly stream errors upwards. All calls made to this function in the iomanager are checked for their return value, so errors propagate correctly upwards. The address it receives is relative to the beginning of the disc, no assumptions about `buf` may be made, it can be withing ioman's cache memory range, but it could also be a buffer from userspace. The function will return 0 on success and -1 on failure.	
`ioman_writeSector`	`esint8 (IOManager *ioman, euint32 address, euint8* buf)`
This function does the actual writing to the hardware, it is the one and only function that calls `if_writeBuf()` , here a retry on failure policy could be implemented. This function will correctly stream errors upwards. All calls made to this function in the iomanager are checked for their return value, so errors propagate correctly upwards. The address it receives is relative to the beginning of the disc, no assumptions about `buf` may be made, it can be withing ioman's cache memory range, but it could also be a buffer from userspace. The function will return 0 on success and -1 on failure.	
`ioman_getSector`	`euint8* (IOManager *ioman,euint32 address, euint8 mode)`

<table>
<tr><th colspan="2">I/O Manager Functions (continued)</th></tr>
<tr><td colspan="2">This function is the one that is called most from the higher library routines. It is the function that will present you with a pointer to memory containing sector number `address` . There are several modes that you can select or combine.
<table>
<tr><td>IOM_MODE_READONLY</td><td>This attribute says to ioman that it needs a buffer only for reading. This does not mean that you are allowed to write to it, doing so results in undefined behavior. You cannot combine this option with the `IOM_MODE_READWRITE` option.</td></tr>
<tr><td>IOM_MODE_READWRITE</td><td>This attribute says to ioman that it would like not only to read from but also to write to that buffer. When you release the sector your changes will be written to disc. This may not happen immediately though, if you want to force it take a look at the `ioman_flushRange()` function. This option cannot be combined with the `IOM_MODE_READONLY` option.</td></tr>
<tr><td>IOM_MODE_EXP_REQ</td><td>This option tell the iomanager that the request is exceptional, for example that the request is unlikely to happen again. The library adds this flags to the options when requesting the bootrecord, to prevent it from getting a high rating, which should prevent it from being removed from the cache.</td></tr>
</table>
These options can be combined by ORing them together.</td></tr>
<tr><td>ioman_releaseSector</td><td>esint8 (IOManager *ioman,euint8* buf)</td></tr>
<tr><td colspan="2">This function tells ioman that you are done with one of the cache elements and that it can do it's bidding with it. Forgetting to call this function may result in deadlocked iomanagers.</td></tr>
<tr><td>ioman_directSectorRead</td><td>esint8 (IOManager *ioman,euint32 address, euint8* buf)</td></tr>
<tr><td colspan="2">This is a variant of the normal getsector. Sometimes you need a sector from the disc, but all you want to do with it is export it directly to userbuffers. It would be foolish to force a caching of that sector if there is external space available for it.
This function will fetch sector `address` from disc and place it in the memory pointed to by `buf` . Should there be a free spot available the sector will be cached there, so that it may be used in the future. If the sector was available from cache in the first place, it will simply be `memCpy()` 'd from the cache to the userspace buffer.</td></tr>
<tr><td>ioman_directSectorWrite</td><td>esint8 (IOManager *ioman,euint32 address, euint8* buf)</td></tr>
<tr><td colspan="2">This function is based on the same philosophy as `ioman_directSectorRead()` , however, contrary to what the name may lead to believe it also passes through a caching layer. If there is an unused spot (or the sector is in cache), the userbuffer will be copied to that spot and will remain there until the space is needed or a flush is forced.</td></tr>
<tr><td>ioman_flushRange</td><td>esint8 (IOManager *ioman,euint32 address_low, euint32 address_high)</td></tr>
</table>

I/O Manager Functions (continued)	
This function is used to ask ioman to flush all sectors to disc that are in a specific range. For example you might want to flush a specific range of your filesystem without needlessly disturb other parts. The range is `address_low <= n => address_high` . Off course only sectors that are marked as writable are flushed to disc.	
`ioman_flushAll`	`esint8 (IOManager *ioman)`
This function will cause ioman to flush out all cache units that are marked writable. If they do not have any users, they will lose their writable mark.	

5.5 C library for EFSL

This section of the manual describes the minimalistic C library functions that were created for EFSL. Since EFSL was designed for ultimate portability, no assumptions about the workings or even the presence of a C library could be made. Fortunately only very few functions had to be created that mimicked the operations of well known C library functions.

PLibC Functions	
`strMatch`	`euint16 strMatch(eint8* bufa, eint8*bufb,euint32 n)`
This function compares the strings `bufa` and `bufb` for `n` bytes. It will return the number of bytes in that section that does not match. So if you want to compare two strings the return value should be 0, for the strings to match over the entire `n` area.	
`memCpy`	`void memCpy(void* psrc, void* pdest, euint32 size)`
This function will copy the contents at location `psrc` to location `pdest` over a range of `size` bytes.	
`memClr`	`void memClr(void *pdest,euint32 size)`
This function will set the memory at `pdest` to value `0x00` for a range of `size` bytes.	
`memSet`	`void memSet(void *pdest,euint32 size,euint8 data)`
This function will set the memory at `pdest` to value `data` for a range of `size` bytes.	

6 Legal notes

This library is subject to the Lesser General Public License version 2.1. We have chosen this license in stead of the BSD license because we feel strongly that more effort was needed in the field of quality software in the embedded field.

Please note that if you make changes to the library itself, those modifications must be made public, but that writing support for new hardware and linking it into the library, does not fall under this category. However, we would off course appreciate it tremendously if you would send us in code to support new hardware.

6.1 GNU Lesser General Public License

distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined

library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Lesser General Public License for more details.
```