EEPROM File System Design Notes

Overview

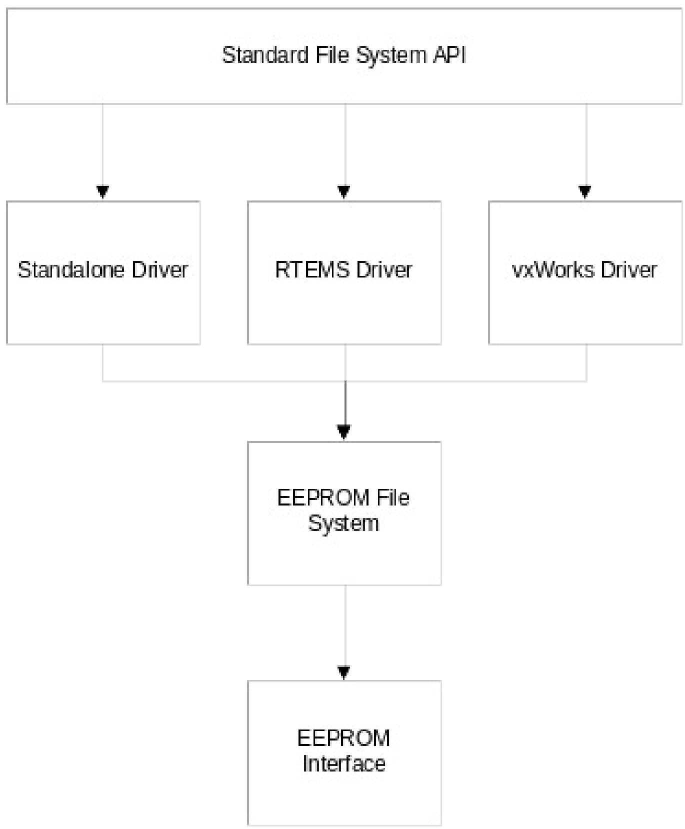
The EEPROM File System is a custom file system designed for data stored in EEPROM. The EEPROM File System is a slot based file system where each slot is a fixed size contiguous region of memory allocated for a single file. Note that each slot can be a different size. The size of each slot is determined when the file system is created and is based on the size of the file to be stored in the slot. Additional free space can be to the end of each slot to allow room for the file to grow in size if necessary and new files can be created at runtime.

Design Goals

- Easy on-orbit maintenance. Since each file is located in a known fixed contiguous region of memory, it is easy to patch or reload the file from the ground.
- Support for the standard file system api. Looks like any other file system to application software.
- Minimize the number of writes to EEPROM.
- Simple implementation.
- Low memory/system overhead.
-
- The EEFS shall store a single file in a fixed continuous region of memory. This is opposed to a block device where a single file may be broken up into multiple non-continuous memory blocks
- The EEFS shall support a standard file system api.
- Patching
- Disadvantages to this approach vs a block device

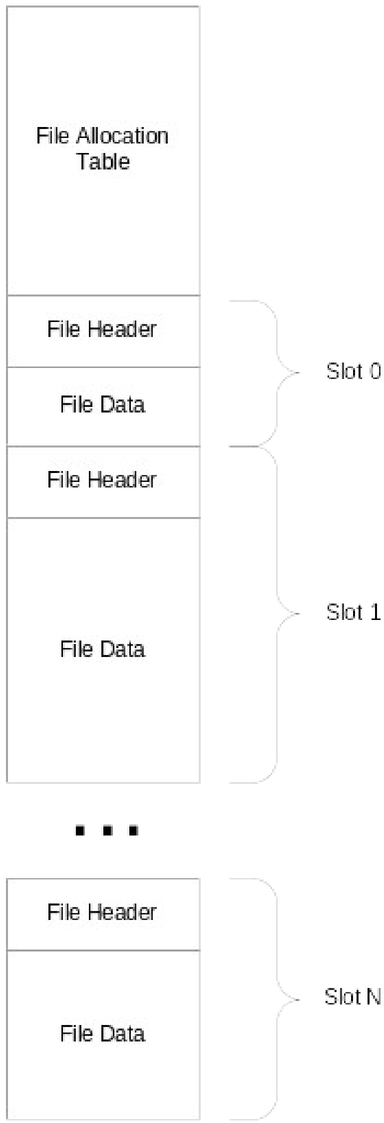
Layered Architecture

The EEPROM File System is split into four layers. At the top most layer the EEPROM File System supports the standard file system api, which makes it look like any other file system to application software. Each implementation of the file system has an OS Specific Driver layer that maps the standard file system api function calls to EEPROM File System specific function calls. The EEPROM File System layer contains the low level EEPROM File System software, and finally the EEPROM Interface layer includes functions that talk directly to the EEPROM hardware.



File System Structure

The EEPROM File System structure is outlined below. The file system contains a File Allocation Table followed by slots for each file in the file system. Each slot contains a File Header and File Data.



File Allocation Table:

The File Allocation Table defines where in memory each slot starts as well as the maximum slot size for each file. This table is a fixed size regardless of how many files actually reside in the file system and never changes unless a new file is added to the file system using the EEFS_LibCreat() function or the whole file system is reloaded. The maximum number of files that can be added to the file system is determined at compile time by the EEFS_MAX_FILES define. It is important to choose this number carefully since a code patch would be required to change it. The file offsets for each file defined in the File Allocation Table are relative offsets from the beginning of the file system and point to the start of the File Header. Since they are relative offsets the file system in not tied to any physical address. In fact the exact same file system image can be burned into multiple locations of EEPROM and then mounted as different volumes. The size of each slot is specified in bytes and does NOT include the File Header. So each slot actually occupies (File Header Size + Slot Size) bytes of EEPROM.

```
typedef struct
{
    uint32                          Crc;
    uint32                          Magic;
    uint32                          Version;
    uint32                          FreeMemoryOffset;
    uint32                          FreeMemorySize;
    uint32                          NumberOfFiles;
} EEFS_FileAllocationTableHeader_t;

typedef struct
{
    uint32                          FileHeaderOffset;
    uint32                          MaxFileSize;
} EEFS_FileAllocationTableEntry_t ;

typedef struct
{
    EEFS_FileAllocationTableHeader_t    Header;
    EEFS_FileAllocationTableEntry_t     File[EEFS_MAX_FILES];
} EEFS_FileAllocationTable_t;
```

File Header:

Each slot in the file system starts with a File Header. The File Header contains information about the file contained in the slot such as its filename. The File Header is initialized to all 0's if the slot is unused or the file has been deleted.

```
typedef struct
{
    uint32                          Crc;
    uint32                          InUse;  /* FALSE then the file has been deleted */
    uint32                          Attributes;
    uint32                          FileSize;
    time_t                          ModificationDate;
    time_t                          CreationDate;
    char                            Filename[EEFS_MAX_FILENAME_SIZE];
} EEFS_FileHeader_t;
```

File Data:

File Data starts immediately following the File Header and may or may not use all of the available space in the slot.

Inode Table

The Inode Table is a ram table that is used by the file system api to access the file system and is similar in structure to the File Allocation Table. The Inode table is initialized when the function EEFS_LibInitFS() is called and once the Inode table is initialized the File Allocation Table is no longer used. One important difference between the File Allocation Table and the Inode Table is that the Inode Table contains physical address pointers to the start of each file instead of relative offsets. Note also that the Inode table does not cache any information about the file in ram. This means that the file could be patched or reloaded to EEPROM without the need to patch the Inode Table, i.e. the file updates are available to the file system immediately. A disadvantage to this approach is that each File Header must be read from EEPROM when searching the file system for a specific file, for example when a file is opened.

```
typedef struct
{
    void                                *FileHeaderPointer;
    uint32                               MaxFileSize;
} EEFS_InodeTableEntry_t;

typedef struct
{
    uint32                               BaseAddress;
    void                                *FreeMemoryPointer;
    uint32                               FreeMemorySize;
    uint32                               NumberOfFiles;
    EEFS_InodeTableEntry_t               File[EEFS_MAX_FILES];
} EEFS_InodeTable_t;
```

File Descriptor Table

The File Descriptor Table manages all File Descriptors for the EEPROM File System. There is only one File Descriptor Table that is shared by all EEPROM File System volumes. The maximum number of files that can be open at one time is determined at compile time by the EEFS_MAX_OPEN_FILES define.

```
typedef struct
{
    uint32                               InUse;
    uint32                               Mode;
    void                                *FileHeaderPointer;
    void                                *FileDataPointer;
    uint32                               ByteOffset;
    uint32                               FileSize;
    uint32                               MaxFileSize;
    EEFS_InodeTable_t                   *InodeTable;
    uint32                               InodeIndex;
} EEFS_FileDescriptor_t;

EEFS_FileDescriptor_t                    EEFS_FileDescriptorTable[EEFS_MAX_OPEN_FILES];
```

Directory Descriptor Table

The Directory Descriptor Table manages the Directory Descriptor for the EEPROM File System. There is currently only one Directory Descriptor that is shared by all EEPROM File System volumes.

```
typedef struct
{
    uint32                              InUse;
    uint32                              InodeIndex;
    EEFS_InodeTable_t                  *InodeTable;
} EEFS_DirectoryDescriptor_t;

typedef struct
{
    uint32                              InodeIndex;
    char                                Filename[EEFS_MAX_FILENAME_SIZE];
    uint32                              InUse;
    void                               *FileHeaderPointer;
    uint32                              MaxFileSize;
} EEFS_DirectoryEntry_t;
```

EEPROM Access

The EEPROM File System software never directly reads or writes to EEPROM, instead it uses implementation specific EEPROM interface functions. Since not all EEPROM is memory mapped, some EEPROM implementations may require implementation specific functions for accessing EEPROM. The implementation specific EEPROM interface functions are defined as macros in the file eefs_macros.h. By default these macros are defined to use memcpy. Note also that the EEPROM interface functions are protected from shared access by the EEPROM File System however there is nothing that prevents other processes from calling the EEPROM interface functions from outside of the EEPROM File System.

```
#define EEFS_LibEEPROMWrite(Dest, Src, Length) memcpy(Dest, Src, Length)
#define EEFS_LibEEPROMRead(Dest, Src, Length) memcpy(Dest, Src, Length)
#define EEFS_LibEEPROMFlush
```

EEPROM Write Protection

The EEPROM File System can be write protected. This feature is implemented through an implementation specific interface function that returns the write protection state of the file system. This interface function is defined as a macro in the file eefs_macros.h. If the file system is read-only then the macro will be defined to TRUE. If the file system is always write enabled then the macro will be defined to FALSE. If the eeprom has an external write protection interface then a custom function can be called to determine the write protect status. By default this macro is defined to FALSE which means that the file system is not write protected.

```
#define EEFS_LibIsWriteProtected EEPROM_IsWriteProtected(InodeTable->BaseAddress)
```

Mutual Exclusion

Mutual exclusion is implemented by the functions EEFS_LibLock and EEFS_LibUnlock. Since the EEPROM File System is not intended to be used very often and to keep things simple it was decided to implement a single locking mechanism that is shared by all EEPROM File System volumes. This locking mechanism simply locks the shared resource at the start of each function and unlocks the shared resource at the end of each function. It is recommended that semaphores be used as the locking mechanism vs disabling interrupts. Note that since the shared resource is locked for all lower level functions, lower level functions should not be called recursively. The implementation of the EEFS_LibLock and EEFS_LibUnlock functions are defined as macros in the file eefs_macros.h.

```
#define EEFS_LIB_LOCK semTake(EEFS_semId, WAIT_FOREVER);
#define EEFS_LIB_UNLOCK semGive(EEFS_semId);
```

Time Stamps

Time Stamps are implemented by the function EEFS_LibTime. Time stamps are based on the standard library time_t. The implementation of the EEFS_LibTime function is defined as a macro in the file eefs_macros.h. By default this macro is defined to use the standard library time function.

File Attributes

The only File Attribute supported by the EEPROM File System is the Read Only file attribute. This Attribute can be set on a per file basis when the file system is created or it can be changed at run time by calling the function EEFS_LibSetFileAttributes(EEFS_InodeTable_t *InodeTable, char *Filename, uint32 Attributes).

Directories

The EEPROM File System is a flat file system therefore it only supports a single top level directory and does not support sub directories. If you want to group files separately then they should be placed in different volumes. For example /EEFS0_Apps for apps, /EEFS0_Tables for tables etc...

CRC's

The EEPROM File System includes a crc in the File Allocation Table and a crc in the File Header for each file. Currently the crc included in the File Allocation Table is calculated across the entire file system, including unused space (i.e. MaxEepromSize), and is currently only used by bootstrap code to verify the integrity of the file system at boot time. This crc is NOT automatically updated by the file system when files are modified, so this must be done manually. The crc included in each File Header is currently not used by the file system.

Rewriting Existing Files

Existing files can be updated or completely rewritten simply by opening an existing file for write access. The only restriction is that existing files are limited in size to the max file size for that slot. If the file grows larger than the max file size then file write calls will return 0 bytes written.

Creating New Files

New files can be created at runtime by opening a file for write access that does not already exist with the O_CREAT flag or by calling the creat function . When a new file is created a new slot is added to the end of the file system that initially has a max file size equal to all available unused space in the file system. This is done because we won't know the final size of the file until it is closed. When the file is closed the slot is resized to the actual size of the file plus a fixed amount of spare and the additional unused space is returned to the file system. The fixed amount of spare is determined at compile time by the EEFS_DEFAULT_CREAT_SPARE_BYTES define. New files can be added to the file system up to EEFS_MAX_FILES or all available space in the file system is used. Note also that only one new file can be created at a time.

Deleting Files

The file system supports deleting files however it is not recommended. Deleting a file renders the slot containing the file unusable. Note that because of this design the file system will not reclaim space from a deleted file.

Patching Files

Existing files can be easily patched outside of the file system api when necessary. In some cases only the file data that is changing needs to be loaded, however if the file size or the file name is changing then the file header will also have to be updated. File metadata is not cached in ram so file updates only need to occur to eeprom, no other patches are necessary. The address of each slot can be found by looking at the file system map file created by the geneepronfs tool. See the section on Building a File System Image for more information.

Micro EEPROM File System

The Micro version of the EEPROM file system allows bootstrap code access to files in an EEPROM File System. The full implementation of the EEPROM File System is too large to be used in bootstrap code so a simple single function version was developed that returns the starting address of a file given its filename. The bootstrap code can then boot the system from a kernel image that is contained in an EEPROM File System. This software is designed to use very little memory, and is independent of the file system size (i.e. maximum number of files). This means that the bootstrap image will NOT have to be updated whenever the size of the EEPROM File System changes.

Building a File System Image

EEPROM File System images are created using the geneepromfs tool. This command line tool reads an input file that describes the files that will be included in the file system and outputs an EEPROM File System image ready to be burned into EEPROM.

Available Options

```
Build a EEPROM File System Image.

  Options:
  -e, --endian=big or little        set the output encoding (big)
  -s, --eeprom_size=SIZE            set the size of the target eeprom (2 Mb)
  -t, --time=TIME                   set the file timestamps to a fixed value
  -f, --fill_eeprom                 fill unused eeprom with 0's
  -v, --verbose                     print the name of each file added to the
                                      file system
  -m, --map=FILENAME                output a file system memory map
  -V, --version                     output version information and exit
  -h, --help                        output usage information and exit

  The INPUT_FILE is a formatted text file that specifies the files to be added
    to the file system.  Each entry in the INPUT_FILE contains the following
    fields separated by a comma:
    1. Input Filename: The path and name of the file to add to the file system
    2. EEFS Filename: The name of the file in the eeprom file system.  Note the
         EEFS Filename can be different from the original Input Filename
    3. Spare Bytes: The number of spare bytes to add to the end of the file.
         Note also that the max size of the file is rounded up to the nearest
         4 byte boundary.
    4. Attributes: The file attributes, EEFS_ATTRIBUTE_NONE or EEFS_ATTRIBUTE_READONLY.
    Each entry must end with a semicolon.
    Comments can be added to the file by preceding the comment with an
      exclamation point.

    Example:
    !
    ! Input Filename             EEFS Filename     Spare Bytes  Attributes
    !--------------------------------------------------------------------------------
      /../images/cfe-core.slf,   file1.slf,        100,         EEFS_ATTRIBUTE_NONE;
```

Sample Input File

```
! Input Filename              EEFS Filename      Spare Bytes  Attributes
!-------------------------------------------------------------------------------
 startupA.scr,               startupA.scr,           128,         EEFS_ATTRIBUTE_NONE;
 startupB.scr,               startupB.scr,           128,         EEFS_ATTRIBUTE_NONE;
 cfe_es_startup.scr,         cfe_es_startup.scr,     128,         EEFS_ATTRIBUTE_NONE;
 cfe-core.o,                 cfe-core.o,            2048,         EEFS_ATTRIBUTE_NONE;
 cdh_lib.o.gz,               cdh_lib.o.gz,          2048,         EEFS_ATTRIBUTE_NONE;
 cfs_lib.o.gz,               cfs_lib.o.gz,          2048,         EEFS_ATTRIBUTE_NONE;
 sw.o.gz,                    sw.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 sw_a_netwtbl.tbl,           sw_a_netwtbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 sw_a_sca_rttbl.tbl,         sw_a_sca_rttbl.tbl,     128,         EEFS_ATTRIBUTE_NONE;
 sw_a_scb_rttbl.tbl,         sw_a_scb_rttbl.tbl,     128,         EEFS_ATTRIBUTE_NONE;
 sw_a_transtbl.tbl,          sw_a_transtbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 sw_b_netwtbl.tbl,           sw_b_netwtbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 sw_b_sca_rttbl.tbl,         sw_b_sca_rttbl.tbl,     128,         EEFS_ATTRIBUTE_NONE;
 sw_b_scb_rttbl.tbl,         sw_b_scb_rttbl.tbl,     128,         EEFS_ATTRIBUTE_NONE;
 sw_b_transtbl.tbl,          sw_b_transtbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 sw_sbccnfgtbl.tbl,          sw_sbccnfgtbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 sw_scommacfgtbl.tbl,        sw_scommacfgtbl.tbl,    128,         EEFS_ATTRIBUTE_NONE;
 sw_scommbcfgtbl.tbl,        sw_scommbcfgtbl.tbl,    128,         EEFS_ATTRIBUTE_NONE;
 xb.o.gz,                    xb.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 xb_cchntblin.tbl,           xb_cchntblin.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 xb_cchntblsc.tbl,           xb_cchntblsc.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 xb_cdsctblin.tbl,           xb_cdsctblin.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 xb_cdsctblsc.tbl,           xb_cdsctblsc.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 xb_ttbl.tbl,                xb_ttbl.tbl,            128,         EEFS_ATTRIBUTE_NONE;
 ci.o.gz,                    ci.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 ci_fec_keytbl.tbl,          ci_fec_keytbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 gnc.o.gz,                   gnc.o.gz,              2048,         EEFS_ATTRIBUTE_NONE;
 cf.o.gz,                    cf.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 cf_cfgtable.tbl,            cf_cfgtable.tbl,        128,         EEFS_ATTRIBUTE_NONE;
 cs.o.gz,                    cs.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 cs_eepromtbl.tbl,           cs_eepromtbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 cg.o.gz,                    cg.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 cg_mtb_tbl.tbl,             cg_mtb_tbl.tbl,         128,         EEFS_ATTRIBUTE_NONE;
 cg_mtb_b_tbl.tbl,           cg_mtb_b_tbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 cg_rw_tbl.tbl,              cg_rw_tbl.tbl,          128,         EEFS_ATTRIBUTE_NONE;
 cg_rw_f0_tbl.tbl,           cg_rw_f0_tbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 cg_rw_f1_tbl.tbl,           cg_rw_f1_tbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 cg_rw_f2_tbl.tbl,           cg_rw_f2_tbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 cg_rw_f3_tbl.tbl,           cg_rw_f3_tbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 cg_rw_f4_tbl.tbl,           cg_rw_f4_tbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
 cg_sa_tbl.tbl,              cg_sa_tbl.tbl,          128,         EEFS_ATTRIBUTE_NONE;
 cg_hga_tbl.tbl,             cg_hga_tbl.tbl,         128,         EEFS_ATTRIBUTE_NONE;
 cg_prop_tbl.tbl,            cg_prop_tbl.tbl,        128,         EEFS_ATTRIBUTE_NONE;
 cl.o.gz,                    cl.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 cl_rnmode_tbl.tbl,          cl_rnmode_tbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 cl_spmode_tbl.tbl,          cl_spmode_tbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 cl_gspmode_tbl.tbl,         cl_gspmode_tbl.tbl,     128,         EEFS_ATTRIBUTE_NONE;
 cl_msmode_tbl.tbl,          cl_msmode_tbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 cl_slew_tbl.tbl,            cl_slew_tbl.tbl,        128,         EEFS_ATTRIBUTE_NONE;
 cl_dhmode_tbl.tbl,          cl_dhmode_tbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 cl_dvmode_tbl.tbl,          cl_dvmode_tbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 cl_modemgr_tbl.tbl,         cl_modemgr_tbl.tbl,     128,         EEFS_ATTRIBUTE_NONE;
 cl_hga_tbl.tbl,             cl_hga_tbl.tbl,         128,         EEFS_ATTRIBUTE_NONE;
 cl_sa_tbl.tbl,              cl_sa_tbl.tbl,          128,         EEFS_ATTRIBUTE_NONE;
 cl_system_tbl.tbl,          cl_system_tbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 cl_target_tbl.tbl,          cl_target_tbl.tbl,      128,         EEFS_ATTRIBUTE_NONE;
 ds.o.gz,                    ds.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 ds_file_tbl.tbl,            ds_file_tbl.tbl,        128,         EEFS_ATTRIBUTE_NONE;
 DS_LEO_filter_tbl.tbl,      DS_LEO_filter_tbl.tbl,  128,         EEFS_ATTRIBUTE_NONE;
 DS_DV_filter_tbl.tbl,       DS_DV_filter_tbl.tbl,   128,         EEFS_ATTRIBUTE_NONE;
 DS_Nom_filter_tbl.tbl,      DS_Nom_filter_tbl.tbl,  128,         EEFS_ATTRIBUTE_NONE;
 DS_Safe_filter_tbl.tbl,     DS_Safe_filter_tbl.tbl, 128,         EEFS_ATTRIBUTE_NONE;
 di.o.gz,                    di.o.gz,               2048,         EEFS_ATTRIBUTE_NONE;
 di_mtb_b_tbl.tbl,           di_mtb_b_tbl.tbl,       128,         EEFS_ATTRIBUTE_NONE;
```

```
di_css_tbl.tbl,        di_css_tbl.tbl,        128,        EEFS_ATTRIBUTE_NONE;
```

Sample Map File

Offset	Size	Section	Slot	Filename	File Size	Spare	Max Size	Crc	Attributes
0	4120	FAT							
4120	64	Header	0						
4184	288	Data	0	startupA.scr	160	128	288	0xFFFF9818	0
4472	64	Header	1						
4536	288	Data	1	startupB.scr	160	128	288	0xFFFF9888	0
4824	64	Header	2						
4888	4112	Data	2	cfe_es_startup.scr	3984	128	4112	0x00005A51	0
9000	64	Header	3						
9064	333056	Data	3	cfe-core.o	331007	2049	333056	0x0000713B	0
342120	64	Header	4						
342184	26480	Data	4	cdh_lib.o.gz	24429	2051	26480	0xFFFFB9B0	0
368664	64	Header	5						
368728	3580	Data	5	cfs_lib.o.gz	1529	2051	3580	0xFFFFB62B	0
372308	64	Header	6						
372372	23256	Data	6	sw.o.gz	21206	2050	23256	0xFFFF8ACC	0
395628	64	Header	7						
395692	308	Data	7	sw_a_netwtbl.tbl	180	128	308	0xFFFFF6E1	0
396000	64	Header	8						
396064	396	Data	8	sw_a_sca_rttbl.tbl	266	130	396	0xFFFFD221	0
396460	64	Header	9						
396524	396	Data	9	sw_a_scb_rttbl.tbl	266	130	396	0xFFFFB2E4	0
396920	64	Header	10						
396984	724	Data	10	sw_a_transtbl.tbl	596	128	724	0x00002576	0
397708	64	Header	11						
397772	308	Data	11	sw_b_netwtbl.tbl	180	128	308	0x0000659C	0
398080	64	Header	12						
398144	396	Data	12	sw_b_sca_rttbl.tbl	266	130	396	0xFFFFAADB	0
398540	64	Header	13						
398604	396	Data	13	sw_b_scb_rttbl.tbl	266	130	396	0xFFFFCA1E	0
399000	64	Header	14						
399064	724	Data	14	sw_b_transtbl.tbl	596	128	724	0xFFFFABBD	0
399788	64	Header	15						
399852	332	Data	15	sw_sbccnfgtbl.tbl	204	128	332	0x00002A7C	0
400184	64	Header	16						
400248	332	Data	16	sw_scommacfgtbl.tbl	204	128	332	0xFFFFE74F	0
400580	64	Header	17						
400644	332	Data	17	sw_scommbcfgtbl.tbl	204	128	332	0x0000398F	0
400976	64	Header	18						
401040	30592	Data	18	xb.o.gz	28541	2051	30592	0xFFFFFDE1	0
431632	64	Header	19						
431696	1284	Data	19	xb_cchntblin.tbl	1156	128	1284	0x00002737	0
432980	64	Header	20						
433044	1284	Data	20	xb_cchntblsc.tbl	1156	128	1284	0xFFFFBADC	0
434328	64	Header	21						
434392	844	Data	21	xb_cdsctblin.tbl	716	128	844	0x00003F3C	0
435236	64	Header	22						
435300	844	Data	22	xb_cdsctblsc.tbl	716	128	844	0x000017C4	0
436144	64	Header	23						
436208	1344	Data	23	xb_ttbl.tbl	1216	128	1344	0x00006318	0
437552	64	Header	24						
437616	16496	Data	24	ci.o.gz	14447	2049	16496	0xFFFFC696	0
454112	64	Header	25						
454176	8436	Data	25	ci_fec_keytbl.tbl	8308	128	8436	0x00003BA1	0
462612	64	Header	26						
462676	5244	Data	26	gnc.o.gz	3196	2048	5244	0xFFFFB88C	0
467920	64	Header	27						
467984	80340	Data	27	cf.o.gz	78290	2050	80340	0xFFFFC059	0
548324	64	Header	28						
548388	4228	Data	28	cf_cfgtable.tbl	4100	128	4228	0x00002769	0
552616	64	Header	29						
552680	22216	Data	29	cs.o.gz	20165	2051	22216	0x00002DFF	0
574896	64	Header	30						
574960	436	Data	30	cs_eepromtbl.tbl	308	128	436	0xFFFFB43C	0
575396	64	Header	31						
575460	29692	Data	31	cg.o.gz	27641	2051	29692	0xFFFFB484	0

605152	64	Header	32						
605216	492	Data	32	cg_mtb_tbl.tbl	364	128	492	0x00007A7F	0