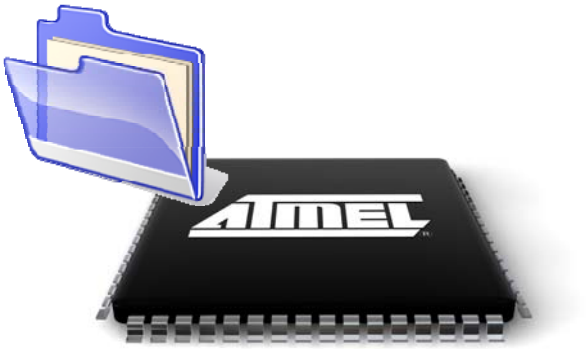
AVR114: Using the ATMEL File System management for AT32UC3x, AT90USBx and ATmega32U4



AVR® Microcontrollers

Application Note

1 Introduction

Atmel provides a File System management for parts AT32UC3x, AT90USBx and ATMega32U4. The File System module is optimized for the AVR core, so the footprint is small (code <15KB, ram <800B) and the speed is high.

The File System Management provides many features such as opening multiple files at the same time, or play list management. This application note will help the reader to use the File System management.

Rev. 7824A-AVR-09/08

2 Features & limitations

2.1 Main Features

- Mount FAT12, FAT16, FAT32
- ASCII and **UNICODE**
- **Multiple files open at the same time (concurrent access)**
- Format FAT12/FAT16/FAT32
- Navigation by string path supported
- Directory create, delete, rename
- File I/O access (**DMA compatible**)
- File create, delete, rename, **copy**

2.2 Plug-IN Features

- **Text file reader** (ASCII, UTF16, UTF16BE, UTF8)
- **Play list file reader** (*.m3u, *.m3u8, *.pls and *.smp)
- Navigation with extension filter
- Navigation in flat mode
- **Automatic Navigation with repeat and random** features
- Posix interface

2.3 Limitations

Limitations from **FAT12/FAT16/FAT32 specifications**:

- disk size supported up to 2TB
- file size supported up to 4GB
- up to 512 *file entries** supported in root directory of FAT12/16
- up to 65535 *file entries** supported in a directory
- No folder depth limitation

**A file can use more than one file entry, depending on the length of its name.*

2.4 Implementation Limitations

- No FAT integrity check at startup
- File list order is the file creation order (no sort by file name supported)
- Supports only a FAT area number of 2 (it is still highly recommended by Microsoft that no value other than 2 should be used.).

3 Overview

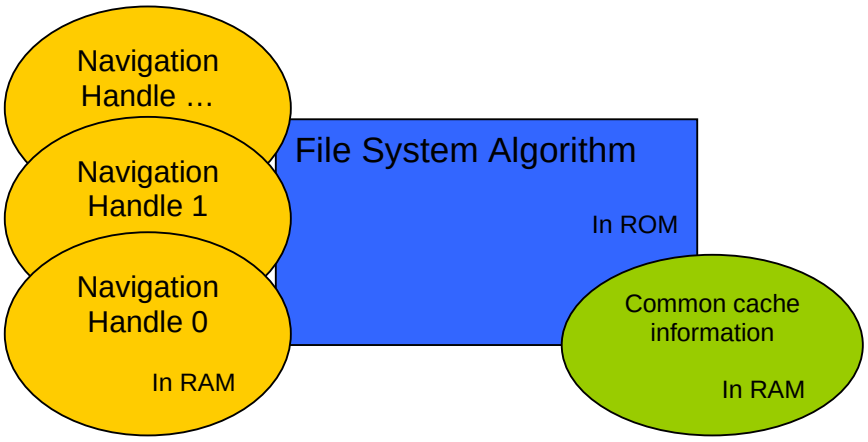
3.1 Introduction

The Atmel FileSystem uses a main notion the *Navigator handle*.

A ***navigation handle*** allows to explore one directory or to open one file. The system supports many handles to authorize the multi-exploration and multi-opening of files simultaneously.

The navigation handle is very small and requires only 40B of RAM.

Figure 3-1. Module organization



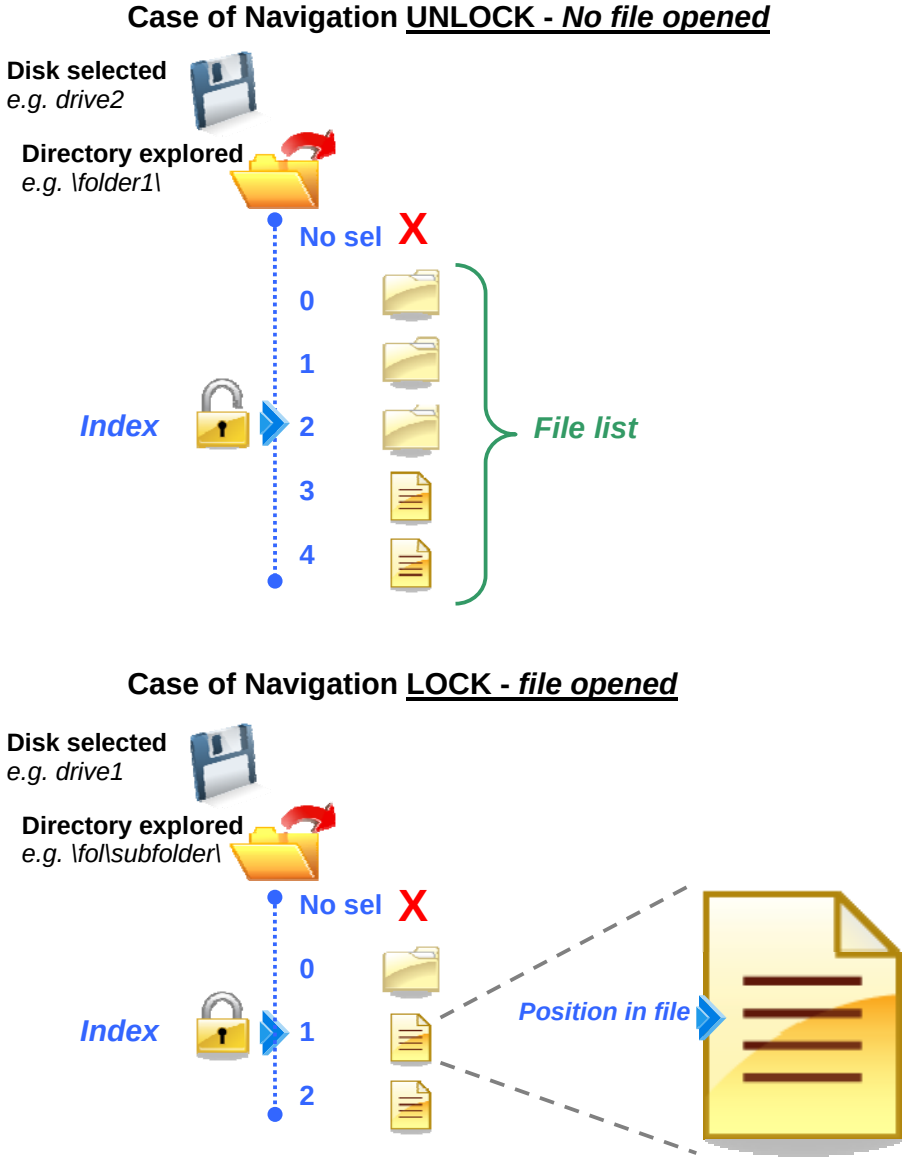
3.2 Navigation

File access by string path is possible but the *navigation handle* allows accessing a list of files. The navigation explores a directory contents, which is called ***file list***, with simples commands like *next/previous* used to move a index in the list. The *file list* can include directories and files. When the file selected by the index is opened then the navigation is locked. See figure 4.2.

The *navigation handle* (or *navigation ID*) contains the following information:

- The disk number
- The reference of the directory which is explored
- The index position in the *file list*
- A lock/unlock navigation flag (file opened/not opened)
- A position in the file when the file is opened

Figure 3-2. Navigation organization

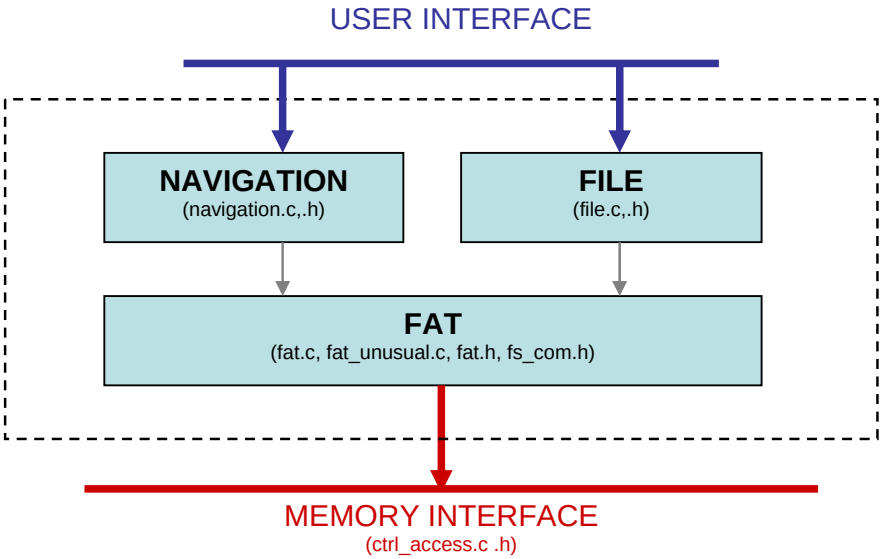


When a disk is mounted, the default directory explored is the root directory. You can then move the index to a subfolder and select it (*"cd folder"* command) as the new directory being explored. Also, you can select the parent directory (*"cd.."* command) as the new directory being explored.

4 Architecture

4.1 Core

Figure 4-1. Core architecture



4.1.1 FAT

The FAT module is the low level module and decodes the FAT structure. All routines of the module are private and cannot be called by a user.

4.1.2 Navigation

This module provides the routines:

- to select a navigator handle
- to navigate in the *file list*
- to change the selected directory
- to modify the directory tree
- to get information about disks/directories/files

All routines are described in *navigation.h* file.

4.1.3 File

This module provides the file I/O control:

- One byte transfer, *file_getc() - file_putc()* **slow**
- RAM Buffer transfer, *file_read_buf() - file_write_buf()* ↓
- Direct transfer*, *file_read() - file_write()* **fast**

*The Direct transfer is developed to transfer file data directly between two memory areas without transferring the data in the RAM space (e.g. DMA between two memory spaces).

All routines are described in *file.h* file.

4.1.4 Memory

The interfaces with the memory are:

- *mem_test_unit_ready()*, to check the memory state
- *mem_wr_protect()*, to check the memory write protection
- *memory_2_ram()*, to read the memory
- *ram_2_memory()*, to write in the memory

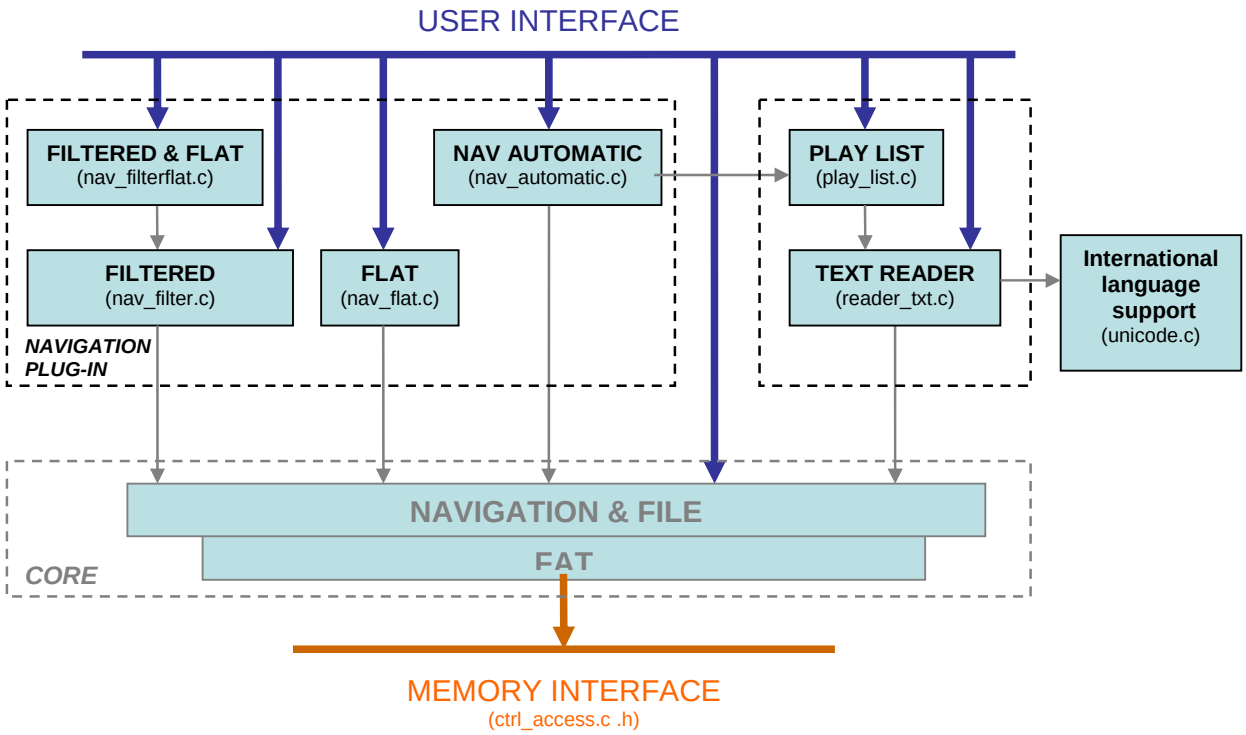
All routines are described in *ctrl_access.h* file.

4.1.5 Error control

Many routines return a status *TRUE* or *FALSE*. In case of FALSE status, the *fs_g_status* global variable contains the error identifier in order to retrieve more information about the error. The error list is available in *fs_com.h* file.

4.2 Plug-In

Figure 4-2. Plug-In architecture



4.2.1 Text reader

This plug-in allows opening a basic text file in read only mode.

The supported text formats are ASCII, UTF16, UTF16BE, and UTF8.

The multi open text file is supported.

4.2.2 Play list

This plug-in manages a file play list in read only mode.

The supported extensions are *.m3u, *.m3u8, *.pls and *.smp.

The play list size limitation is 65535 files.

Multiple opening is not supported.

4.2.3 Automatic Navigation

Automatic navigation has been developed for a player/viewer module.

The plug-in builds a "file list" with the following user specifications:

- Extension filter
- Limitation scan (folder, folder and sub folder, one disk, all disks)
- Random feature

4.2.4 Filtered mode

This plug-in filters the “file list” by extension from the navigation module.

Example:

```
// A disk architecture
folder1
|  folder3
|  |  file4.mp3
|  file5.txt
folder2
|  file6.txt
file1.mp3
file2.txt
file3.mp3

// "File list" provided by navigator
// "root" is the "directory selected"
folder1
folder2
file1.mp3
file2.txt
file3.mp3

// "File list" provided by navigator
// "folder1" is the "directory selected"
folder3
file5.txt

// "File list" provided by plug-IN “FILTER navigator” initialized with
“*.mp3” filter
// "root" is the "directory selected"
folder1
folder2
file1.mp3
file3.mp3

// "File list" provided by plug-IN “FILTER navigator” initialized with
“*.mp3” filter
// "folder1" is the "directory selected"
folder3
```

4.2.5 Flat mode

The FLAT mode ignores the folder level and provides a “file list” with all files/folders present in sub folder of folder selected.

Example:

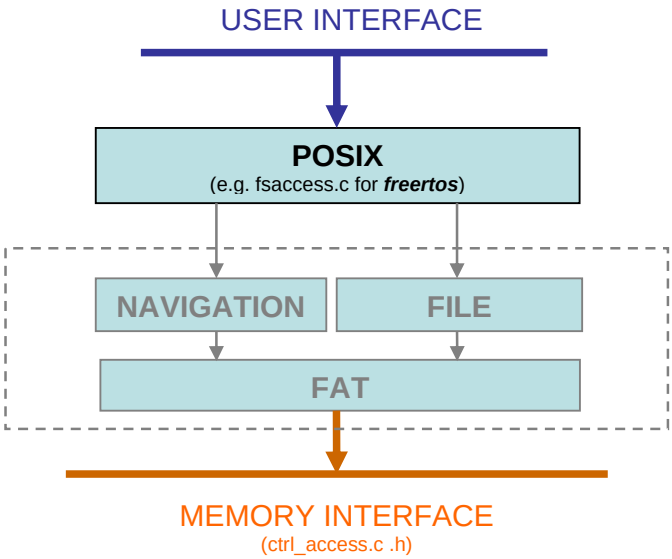
```
// A disk architecture
folder1
|  folder3
|  |  file4
|  file5
folder2
|  file6
file1
file2
file3

// "File list" provided by navigator
// "root" is the "directory selected"
folder1
folder2
file1
file2
file3

// "File list" provided by navigator
// "folder1" is the "directory selected"
folder3
file5

// "File list" provided by plug-IN “FLAT navigator”
// "root" is the "directory selected"
folder1
folder3
file4
file5
folder2
file6
file1
file2
file3

// "File list" provided by plug-IN “FLAT navigator”
// "folder1" is the "directory selected"
folder3
file4
file5
```

4.2.6 Filtered & Flat mode

This plug-in includes the features of “Filtered mode" and "Flat mode".

4.3 POSIX Interface

Figure 4-3. POSIX architecture



***POSIX:** Portable Operating System Interface compatible with variants of the Unix operating system.

This interface is only available on AVR32 UC3 product.

5 Configuration

5.1 Core

Depending on your application, you can optimize the core footprint with the following configuration:

- Enable/Disable FATs supported (FAT12/FAT16/FAT32)
- Enable/Disable UNICODE or ASCII support
- Select a feature level (READ / WRITE / WRITE_COMPLET*)
- Select the cache information size
- Select the maximal number of navigators handles.

**WRITE_COMPLET includes WRITE features and a few additional ones.*

The configuration is defined in ***conf_explorer.h*** file :

```
// Include header file to provide the memory routines (e.g. memset(), memcpy_ram2ram() ...)
#define LIB_MEM              <string.h>
// Include header file to provide the disk access routines (e.g. ram_2_memory(), mem_wr_protect() ...)
#define LIB_CTRLACCESS       "ctrl_access.h"

// Supported FATs (ENABLED or DISABLED).
#define FS_FAT_12            ENABLED
#define FS_FAT_16            ENABLED
#define FS_FAT_32            ENABLED

// The explorer may support either the ASCII or the UNICODE string format, or both.
#define FS_ASCII             DISABLED
#define FS_UNICODE           ENABLED

// The navigator may support only the first partition (DISABLED), or multiple partitions (ENABLED).
#define FS_MULTI_PARTITION   DISABLED

// Level of features in File system core.
// Select among:
//  - FSFEATURE_READ:          All read functions.
//  - FSFEATURE_WRITE:         nav_file_copy(), nav_file_paste(), nav_file_del(), file_create(),
file_open(MODE_WRITE), file_write(), file_putc().
//  - FSFEATURE_WRITE_COMPLET:  FSFEATURE_WRITE functions and nav_drive_format(), nav_dir_make(),
nav_file_rename(), nav_file_dateset(), nav_file_attributset().
//  - FSFEATURE_ALL:           All functions.
#define FS_LEVEL_FEATURES    (FSFEATURE_READ | FSFEATURE_WRITE_COMPLET)

// Number of caches used to store a cluster list of files (must be > 0)
// Interesting in case of many `open file'
#define FS_NB_CACHE_CLUSLIST  3

// The number of navigator ID defines the number maximum of file opening in a time, and the number
maximum of exploration in a time (0<n<256). Each navigator uses less than 50B of RAM.
#define FS_NB_NAVIGATOR      3

// The navigator use the following navigator ID to open the `copy file'
// and the current navigator ID to create the `paste file'.
#define FS_NAV_ID_COPYFILE   2 // Must be different of current ID used when the paste routine is
called.
```

5.2 Plug-In

The Plug-In configurations are defined in ***conf_explorer.h*** file.

5.2.1 Play list

```
// Navigator used to open the file playlist
#define FS_NAV_ID_PLAYLIST    2
// Playlist Interface to allocate a space to store the current path included in play list
// Example with alloc library
#define  PLAYLIST_BUF_ALLOC( size )    malloc( size )
#define  PLAYLIST_BUF_FREE( buf )      free( buf )
// Example with no alloc library
#define  PLAYLIST_BUF_ALLOC( size )    ((sizeof(g_buffer_512)>512)? NULL : g_buffer_512)
#define  PLAYLIST_BUF_FREE( buf )
```

5.2.2 Automatic Navigation

```
// The feature "count all files available" may be disabled to save time at startup
#define  FS_NAV_AUTOMATIC_NBFILE  ENABLE
// Specify the file order in the list provided by the plug-in "Automatic Navigation" (nav_automatic.c)
#define  NAV_AUTO_FILE_IN_FIRST     // Uncomment to disable feature
// Size of the split for the random (Unit 8 files)
#define  NAVAUTO_MAX_RANGE_RAND   8     // 8*8= 64 file
// Random value methode (byte value)
#include "rand.h"
#define  NAVAUTO_GET_RAND( value )   (value=rand())
```

5.2.3 Navigation Flat

```
// Allow modules FLAT mode (nav_flat.c & navfilterflat.c)
#define  NAVIGATION_MODE_FLAT       // Uncomment to disable feature
```

6 Examples

Note: All examples assume that the Atmel file system is initialized like the example “Power ON/OFF sequence”.

6.1 Power ON/OFF sequence

Sequences to initialize the File System module:

```
nav_reset();
```

Sequences to execute before stopping the File System module:

```
// If you have opened files then close them
// Flush data eventually present in FAT cache
nav_exit();
```

Norms: It is not authorized to have two File System managements on the same disk at the same time.

If you have a system which shares disks between Atmel File System and another File System, then you must stop the Atmel FS before jumping to the other FS and reinitialize Atmel FS after exiting from the other FS.

Example: if the Atmel µc is in Device MassStorage mode, the Atmel FS must be turned off in order to let the USB host FS (e.g. the Windows® one) take care of the MassStorage device.

6.2 Check the disk state

Two routines are necessary to check the disk:

- *nav_drive_set()* allows to detect the **presence of a memory driver**.
- *nav_partition_mount()* checks the **presence of memory** and tries to **mount a partition**.

Note: The number of memory driver may be dynamic (e.g. a U-Disk can have multiple memory drives).

```
Bool check_disk( U8 lun )
{
   nav_select(0); // Select a navigator ID

   if( !nav_drive_set(lun) )
   {
      printf("Driver memory no available\n");
      return FALSE;
   }
   // Here the memory is selected

   if( !nav_partition_mount() )
   {
      switch( fs_g_status )
      {
         case FS_ERR_HW_NO_PRESENT:
         printf("Disk not present\n");
         break;
         case FS_ERR_HW:
         printf("Disk access error\n");
         break;
         case FS_ERR_NO_FORMAT:
         printf("Disk no formated\n");
         break;
         case FS_ERR_NO_PART:
         printf("No partition available on disk\n");
         break;
         case FS_ERR_NO_SUPPORT_PART:
         printf("Partition no supported\n");
         break;
         default :
         printf("Other system error\n");
         break;
      }
      return FALSE;
   }
   // Here the partition on memory is mounted and the navigator is on root dir

   return TRUE;
}
```

6.3 Change the date of a file

This example changes the creation date and last access date.

```
Bool changedate( void )
{
// Example to modify a file date
   const U8 _MEM_TYPE_SLOW_ date_create[]="2005122512301050"; // Date = 12/25/2005 12h30mn10.5s
   const U8 _MEM_TYPE_SLOW_ date_write[]="2006072319005130";  // Date = 07/23/2006 19h51mn30s

   nav_select(0); // Select navigator ID 0 for this sequence

   if( !nav_drive_set( LUN_ID_NF_DISKMASS ))
      return FALSE;
   if( !nav_partition_mount() )
      return FALSE;
   // Select the first file or directory
   if( !nav_filelist_set( 0 , FS_FIND_NEXT ) )
      return FALSE;
   // Modify the creation date
   if( !nav_file_dateset( date_create , FS_DATE_CREATION) )
      return FALSE;
   // Modify the last access date
   if( !nav_file_dateset( date_write , FS_DATE_LAST_WRITE) )
      return FALSE;

   return TRUE;
}
```

6.4 Use path string

You can use a text path (ASCII or UNICODE) to access a file or path.

nav_setcwd() accepts the following string :

- "name.txt" to search a file or a directory in the current directory
- "nam*" to search the first file or directory with matching name in the current directory
- '\' and '/' are equivalent and supported
- "name\" to search a directory in the current directory and enter in that directory
- "name2.txt" to search a file in current directory
- "\name2.txt" to search a file in root directory
- "./" is supported but it is not mandatory
- "../../" is supported
- "A:\" is supported, 'A' corresponding to disk 0

```
Bool search_path( void )
{
   const _MEM_TYPE_SLOW_ U8 path[]="dir1/file.txt";

   nav_select(0);                         // Choose a free navigator

   // Select a disk and mount it
   nav_drive_set(1);
   nav_partition_mount();
   // HERE the navigator is in root dir

#if( (FS_ASCII == ENABLED) && (FS_UNICODE == ENABLED) )
   nav_string_ascii();  // Select a ASCII name format
#endif

   // search file "dir1/file.txt" in current dir of disk 1
   if( !nav_setcwd( (FS_STRING)path , TRUE, FALSE ) )
      return FALSE;

   return TRUE;
}
```

6.5 Accelerate multi-write files

6.5.1 Overview

In case of many "file_write_buf() or file_putc()" call (e.g. log file), the execution may be slow because the number of write accesses on the disk is too important to build the FAT table. If you write data several times in a file and you split the write access (e.g. log file), then it may be interesting to build the FAT table of the file using a more efficient code sequence.

Each executed "file_write_buf() or file_putc()" call (e.g. log file) routine requires a reservation in the FAT table. See Example A.

To increase efficiency, we suggest to create the reservation of all necessary space in the FAT within a single pre-sequence, before other commands are executed. See Example B. This example will result in a more efficient write routine. See section 6.5.4.

6.5.2 Example A

This example is the **usual** sequence to fill a file:

```
// File fill >1MB
#define  FILL_FILE_NB_WRITE   855L
#define  FILL_FILE_BUF_SIZE   120L

Bool fill_file( void )
{
   const UNICODE _MEM_TYPE_SLOW_ name[50]={'l','o','g','.','b','i','n',0};
   U16 u16_nb_write;

   memset( g_trans_buffer , 0x55 , FILL_FILE_BUF_SIZE );

   if( !nav_drive_set(LUN_DISK))    // Enter in disk
      return FALSE;
   if( !nav_partition_mount())      // Mount partition of disk
      return FALSE;

   if( !nav_file_create( (const FS_STRING) name )) // Create file
      return FALSE;
   if( !file_open(FOPEN_MODE_W))    // Open file in write mode with force file size 0
      return FALSE;

   for( u16_nb_write=0; u16_nb_write<FILL_FILE_NB_WRITE; u16_nb_write++ )
   {
      // HERE, at each write file, an allocation in FAT area is run
      // so, if you have many buffers to write the execution may be slow.
      if( !file_write_buf( g_trans_buffer , FILL_FILE_BUF_SIZE ))
      {
         file_close();
         return FALSE;
      }
   }
   file_close();
   return TRUE;
}
```

6.5.3 Example B

This example allows **accelerating write accesses with a preallocated** routine:

```
// File fill >1MB
#define  FILL_FILE_NB_WRITE   855L
#define  FILL_FILE_BUF_SIZE   120L

Bool fill_file_fast( void )
{
   const UNICODE _MEM_TYPE_SLOW_ name[50]={'l','o','g','_','f','a','s','t','.','b','i','n',0};
   _MEM_TYPE_SLOW_ Fs_file_segment g_recorder_seg;
   U16 u16_nb_write;

   memset( g_trans_buffer , 0x55 , FILL_FILE_BUF_SIZE );

   if( !nav_drive_set(LUN_DISK))    // Enter in disk
      return FALSE;
   if( !nav_partition_mount())      // Mount partition of disk
      return FALSE;

   if( !nav_file_create( (const FS_STRING) name )) // Create file
      return FALSE;
   if( !file_open(FOPEN_MODE_W))    // Open file in write mode and forces the file size to 0
      return FALSE;

   // Define the size of segment to prealloc (unit 512B)
   // Note: you can alloc more in case of you don't know total size
   g_recorder_seg.u16_size = (FILL_FILE_NB_WRITE*FILL_FILE_BUF_SIZE + 512L)/512L;

   // ****PREALLLOC****** the segment to fill
   if( !file_write( &g_recorder_seg ))
   {
      file_close();
      return FALSE;
   }

   // Check the size of segment allocated
   if( g_recorder_seg.u16_size < ((FILL_FILE_NB_WRITE*FILL_FILE_BUF_SIZE + 512L)/512L) )
   {
      file_close();
      return FALSE;
   }

   // Close/open file to reset size
   file_close();                    // Closes file. This routine don't remove the previous allocation.
   if( !file_open(FOPEN_MODE_W))    // Opens file in write mode and forces the file size to 0
      return FALSE;

   for( u16_nb_write=0; u16_nb_write<FILL_FILE_NB_WRITE; u16_nb_write++ )
   {
      // HERE, the file cluster list is already allocated and the write routine is faster.
      if( !file_write_buf( g_trans_buffer , FILL_FILE_BUF_SIZE ))
      {
         file_close();
         return FALSE;
      }
   }
   file_close();
   return TRUE;
}
```

6.5.4 Statistics

Results of the following example:

Create a file of 100.2KB (buffer 120B * nb write 855) on a 256MB disk (FAT16-cluster 4KB):

- with **Example A**, the number of write accesses on a same sector in FAT is **50 maximum and 25 average**.
- with **Example B,** the number of write accesses on a same sector in FAT is **2 maximum and 1 average**.

Create a file of 1.1MB (buffer 120B * nb write 10000) on a disk 256MB (FAT16-cluster 4KB):

- with **Example A**, the number of write accesses on a same sector in FAT is **255 maximum and 147 average**.
- with **Example B,** the number of write accesses on a same sector in FAT is **2 maximum and 1 average**.

Note:

It is important to know that FAT time creation depends on the FAT type used. The allocation in FAT is faster on FAT (FAT12,FAT16) than FAT32. If you have a disk size < 2GB then you can force the type of FAT when you call nav_drive_format(FS_FORMAT_FAT). A FAT32 is required to write more (x4) sectors in FAT table than FAT16 for the same file size and same disk size.

6.6 Copy a disk on another disk

This example uses three navigator handles:

- to explore the source disk
- to explore the destination disk
- for the copy/paste feature, navigator defined by FS_NAV_ID_COPYFILE in *conf_explorer.h*

```
Bool copydisk( void )
{
   const UNICODE _MEM_TYPE_SLOW_ name[50];
   U8 u8_folder_level = 0;

//trace("Mount drive\n");
   //** Use three navigators (0 to explore SD, 1 to explore NF disk, 2 used by copy file routine)
   nav_select( 0 );
   if( !nav_drive_set( LUN_ID_MMC_SD ))
      return FALSE;
   if( !nav_partition_mount() )
      return FALSE;
   nav_select( 1 );
   if( !nav_drive_set( LUN_ID_NF_DISKMASS ))
      return FALSE;
   if( !nav_partition_mount() )
      return FALSE;

   // loop to scan and create ALL folders and files
   while(1)
   {
      // No dir in current dir then go to parent dir on SD and NandFlash disk
      while(1)
      {
//trace("Search files or dir\n");
         // Reselect SD
         nav_select( 0 );
         if( nav_filelist_set( 0 , FS_FIND_NEXT ) )
            break;   // a next file and directory is found

         // No other dir or file in current dir then go to parent dir on SD and NandFlash disk
         if( 0 == u8_folder_level )
         {
            // end of update folder
//trace("End of copy\n");
            return TRUE;   //********* END OF COPY **************
         }

//trace("Go to parent\n");
         // Remark, nav_dir_gotoparent() routine go to in parent dir and select the children dir in list
         u8_folder_level--;
         if( !nav_dir_gotoparent() )
            return FALSE;
         // Select NandFlash navigator and go to the same dir of SD
         nav_select( 1 );
         if( !nav_dir_gotoparent() )
            return FALSE;
      } // end of while (1)

      if( nav_file_isdir())
      {
//trace("Dir found - create dir & CD\n");
         //** here, a new directory is found and is selected
         // Get name of current selection (= dir name on SD)
         if( !nav_file_name( (FS_STRING )name , 50  , FS_NAME_GET, FALSE  ))
            return FALSE;
         // Enter in dir (on SD)
         if( !nav_dir_cd())
```

```
            return FALSE;
         u8_folder_level++;
         // Select NandFlash disk
         nav_select( 1 );
         // Create folder in NandFlash disk
         if( !nav_dir_make( (FS_STRING )name ))
         {
            if( FS_ERR_FILE_EXIST != fs_g_status )
               return FALSE;
            // here, error the name exist
         }
         // Here the navigator have selected the folder on NandFlash
         if( !nav_dir_cd())
         {
            if( FS_ERR_NO_DIR == fs_g_status )
            {
               // FYC -> Copy impossible, because a file have the same name of folder
            }
            return FALSE;
         }
         // here, the folder is created and the navigatorS is entered in this dir
      }
      else
      {
//trace("File found - copy file\n");
         //** here, a new file is found and is selected
         // Get name of current selection (= file name on SD)
         if( !nav_file_name( (FS_STRING )name , 50  , FS_NAME_GET , FALSE  ))
            return FALSE;
         if( !nav_file_copy())
            return FALSE;

         // Paste file in current dir of NandFlash disk
         nav_select( 1 );
         while( !nav_file_paste_start( (FS_STRING)name ) )
         {
            // Error
            if( fs_g_status != FS_ERR_FILE_EXIST )
               return FALSE;
//trace("del file\n");
            // File exists then deletes this one
            if( !nav_file_del( TRUE ) )
               return FALSE;
            // here, retry PASTE
         }
         // Copy running
         {
         U8 status;
         do{
            status = nav_file_paste_state(FALSE);
         }while( COPY_BUSY == status );

         if( COPY_FINISH != status )
            return FALSE;
         }
      } // if dir OR file
   } // end of first while(1)
}
```

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.Atmel.com

Technical Support
avr@Atmel.com

Sales Contact
www.Atmel.com/contacts

Literature Request
www.Atmel.com/literature