Small Logger File System

(http://www.tnkernel.com/)

Copyright © 2011 Yuri Tiomkin

Document Disclaimer

The information in this document is subject to change without notice. While the information herein is assumed to be accurate, Yuri Tiomkin (the author) assumes no responsibility for any errors or omissions.
The author makes and you receive no warranties or conditions, express, implied, statutory or in any communications with you. The author specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

Small Logger File System

Copyright © 2011 Yuri Tiomkin
All rights reserved.

Permission to use, copy, modify, and distribute this software in source and binary forms and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

THIS SOFTWARE IS PROVIDED BY THE YURI TIOMKIN AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL YURI TIOMKIN OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.
Brand and product names are trademarks or registered trademarks of their respective holders.

Document Version:

- 1.0

INTRODUCTION

The **SLFS** (**S**mall **L**ogger **F**ile **S**ystem) is a logger file system, implemented on the serial flash with unified sector size (4 KB).

The logger file system uses the FIFO approach - for writing the file is added to the tail of the internal list while for reading the file is taken from the head of the list.
In most cases, after reading, the file is erased from SLFS and disk space is released.
If SLFS disk is full and new file is added, the oldest file(s) will be automatically erased.

Files in SLFS may have an arbitrary length.
SLFS uses the write-ahead journaling for file creating/deleting and wear leveling for the system metadata.

SLFS RAM usage

The SLFS uses global structure DF_FLI (52 bytes), structure DF_DISKINFO (328 bytes) per each disk and the one global buffer with the size of the flash sector (4096 bytes).

SLFS multi-access

The SLFS was designed for the small embedded systems with limited RAM capacity.
In SLFS, the only one file at time may be open - if the file is open for reading or was created for writing, no more files can be open/created until file will be closed.

SLFS Wear leveling

The SLFS has a reserved area N * log (metadata) data size, where N is defined by user.
The SLFS write capacity counter is set to the maximal erasing number value. After each file closing, the counter is decremented. When the capacity counter reaches the zero value, the log (metadata) is relocated to the next position in the reserved area and the capacity counter is set to the maximal erasing number value again.
When there is no more free space available in the reserved area for the metadata relocation, the SLFS is set as read-only.

SLFS API usage

The flash formatting

For the flash formatting, the function ***df_create_mbr()*** is used. Then, one or more SLFS disks may be created by the function ***slfs_create_disk().***

The SLFS initialization

After the system restart, the file system(s) should be initiated by the function ***df_init_disks().***

File writing

File is created for the writing by function ***slfs_file_create().***
The writing is performed by function ***slfs_file_write().***
After finishing writing to the file, the file must be close by the function ***slfs_file_close().***

File reading

File is open for the reading by function ***slfs_file_open().***
The reading is performed by function ***slfs_file_read().***
After finishing reading from the file, the file must be close by the function **slfs_file_close()** or ***slfs_file_close_not_erase()***.

Files enumeration

To get actual number of the files in the SLFS, the function ***slfs_get_files_number()*** should be used**.**
To enumerate all files, the functions ***slfs_enum_files_first()/slfs_enum_files_next()*** are used.

SLFS API FUNCTIONS

df_create_mbr

Create MBR (Master Boot Record)

int df_create_mbr (DF_FLI * fli,
char * name **)**

Parameter:

fli	pointer to the global flash descriptor structure
name	The flash IC name (for example "M25PX64", "MX25L12875" etc.) The name should be zero-terminated. Max name size – 16 bytes, including terminator(0)

Return parameter:

ERR_NO_ERR	Operation completed successfully
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function clears overall flash IC content and prepares the flash for the creating disks with file systems (the two copies of Master Boot Record is created).

Before the function call, the '*fli*' structure fields should contain the information about the flash IC (see an example how to fill the structure fields)

df_create_mbr_disk_entry

Create disk in MBR (Master Boot Record)

int df_create_mbr_disk_entry(**DF_FLI** * fli,
__int32 size_in_sectors,
unsigned __int32 fs_type);

Parameter:

fli	pointer to the global flash descriptor structure
size_in_sectors	the desired space size in sectors (not bytes)
fs_type	- for SLFS file system it should be FS_TYPE_SLFS - to reserve space at the FLASH it should be 0

Return parameter:

ERR_NO_ERR	Operation completed successfully
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function is used to allocate space in flash IC for the file system or to reserve space in the FLASH.

The sequential call of this function will create the reserved areas (logical disks) at flash. Maximum number of disks is DF_MAX_DISKS (4). Each logical disk ('A', 'B', 'C', 'D') may be used for SLFS file system, any other file systems /raw data or just for the reserving.

The desired disk space size in sectors will be automatically updated to fit to the flash size.

df_init_disks

Initiate file systems at flash (after reset)

int df_init_disks(**DF_FLI** * fli)

Parameter:

fli	pointer to the global flash descriptor structure

Return parameter:

ERR_NO_ERR	Operation completed successfully
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

After the system restart, the file system(s) should be initiated by the function call. Before the call, the structure '*fli*' have to be filled (see the function ***init_fli()*** in an example).

slfs_create_disk

Create SLFS disk at the flash

int slfs_create_disk(**DF_FLI** * fli,
int disk_letter)

Parameter:

fli	pointer to the global flash descriptor structure
disk_letter	SLFS disk name. May be 'A','B','C' or 'D'

Return parameter:

ERR_NO_ERR	Operation completed successfully
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function creates the SLFS file system at the *'disk_letter'* logical disk

slfs_file_create

Create file for the writing

```
int slfs_file_create(DF_FLI * fli,
                     SLFS_FILE * hFile,
                     int disk_letter,
                     unsigned char * file_name,
                     __int32 file_size,
                     unsigned long file_attr)
```

Parameter:

fli	pointer to the global flash descriptor structure
hFile	The pointer to the file handler structure. The structure should be allocated before the function call and it will be filled inside the function's operating progress.
disk_letter	The SLFS disk name. May be 'A','B','C' or 'D'
file_name	Filename (zero-terminated). Max size - 16 bytes, including terminator (0). It is used just for the user convenience and it is not used inside SLFS for the file storage. Filenames may be duplicated.
file_size	Total expected file size. It should be defined at the file creation.
file_attr	Attribute of file (32-bit unsigned). It is used just for the user convenience and it is not used inside SLFS for the file storage. File attributes may be duplicated.

Return parameter:

ERR_NO_ERR	Operation completed successfully
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function creates the file for writing only. File will append to the end of internal SLFS files list (FIFO). To open file for the reading, the ***slfs_file_open()*** function should be used.

The total expected file size ('*file_size*' parameter) is defined at the file creation.

The writing data to the file is only sequential (see below).

If overall writing data size will be greater than '*file_size*', the overflow bytes will be cut.

If overall writing data size will be less than '*file_size*', it will be set as error.

After writing all bytes to the file, it must be close with the function ***slfs_file_close().***

slfs_file_write

Write a data to the file

int slfs_file_write(**SLFS_FILE** * hFile,
unsigned char * buf,
__int32 count)

Parameter:

hFile	The pointer to the file handler structure. The structure should be allocated before the function call and it and initialized by the calling ***slfs_file_create()*** function.
buf	Buffer with data to write. The typical buffer size is DF_PAGE_SIZE (256 bytes)
count	Number bytes to write. The *'count*' should be equal DF_PAGE_SIZE (256 bytes); If *'count*' value is less than DF_PAGE_SIZE, it will be set as last bytes in the file and the further writing will be restricted

Return parameter:

Greater than or equal to zero	Number of bytes actually has been write
Less than zero	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

The function provides the data writing to the SLFS file.

The writing is only sequential - the **lseek** operation is not supported.

If overall data size will be greater than '*file_size*' (see ***slfs_file_create()***), the overflow bytes will be cut.

If overall data size will be less than *'file_size'* (see ***slfs_file_create()***), it will be interpreted as error.

slfs_file_open

Open file for the reading

int slfs_file_open(**DF_FLI** * fli,
SLFS_FILE * hFile,
int disk_letter,

Parameter:

fli	pointer to the global flash descriptor structure
hFile	The pointer to the file handler structure. The structure should be allocated before the function call and it will be initialized inside the function's operating progress.
disk_letter	The SLFS disk name. May be 'A','B','C' or 'D'

Return parameter:

ERR_NO_ERR	Operation completed successfully
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

The function opens the first file in the internal SLFS files list (FIFO) for the reading purpose only. To open file for the writing, the ***slfs_file_create()*** function should be used.

The reading is only sequential (see below). After the reading all bytes from the file, it should be close with function ***slfs_file_close()*** or ***slfs_file_close_not_erase()***.

If function ***slfs_file_close()*** is used to close file after reading, the file will be automatically removed from SLFS. The next file will be placed at the head of the internal files list (FIFO), and, accordingly, will be open at next call of the ***slfs_file_open()*** function.

If function ***slfs_file_close_not_erase()*** is used to close file after the reading, the file will not be removed - at the next call of the ***slfs_file_open()*** the same file will be open.

In any case, one of the functions (***slfs_file_close()*** or ***slfs_file_close_not_erase()***) must be invoked to finish the reading operation.

slfs_file_read

Read a data from the file

int slfs_file_read(**SLFS_FILE** * hFile,
unsigned char * buf,
__int32 count)

Parameter:

hFile	The pointer to the file handler structure. The structure should be allocated before the function call and initiated by the calling ***slfs_file_open()*** function.
buf	Buffer for the data to read. The typical buffer size is DF_PAGE_SIZE (256 bytes)
count	Number bytes to read. The *'count*' should be equal DF_PAGE_SIZE (256 bytes); If *'count*' value is less than DF_PAGE_SIZE, it will be set as last bytes to read.

Return parameter:

Greater than or equal to zero	Number of bytes actually has been read
Less than zero	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function provides the reading data from the SLFS file. The reading is only sequential - the **lseek** operation is not supported.

The total file length is available in the field *hFile->fheader.file_len* after file opening (with ***slfs_file_open()*** function.

slfs_file_close

Close file after reading or writing

int slfs_file_close(**SLFS_FILE** * hFile)

Parameter:

hFile	The pointer to the file handler structure

Return parameter:

ERR_NO_ERR	Operation completed successfully
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function may be invoked after reading all bytes from the file at the reading operation and must be invoked after writing all bytes to the file at the writing operation.

For the reading operation only, the file will be automatically removed from SLFS and from the head of the internal SLFS files list (FIFO).

The next file will be placed at the head of the internal files list (FIFO), and, accordingly, will be open at next call of the ***slfs_file_open()*** function.

slfs_file_close_not_erase

Close file after reading without erasing

int slfs_file_close_not_erase(**SLFS_FILE** * hFile)

Parameter:

hFile	The pointer to the file handler structure

Return parameter:

ERR_NO_ERR	Operation completed successfully
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function may be invoked for the file closing after reading all bytes from the file (the reading operation only).

The file '*hFile'* will not be removed from the internal SLFS files list (FIFO) - at the next call of the ***slfs_file_open()*** function the same file will be open.

slfs_enum_files_first

Start file enumeration

int slfs_enum_files_first(**DF_FLI** * fli,
SLFS_FILEFIND * hFileFind,
int disk_letter)

Parameter:

Fli	Pointer to the global flash IC descriptor structure
hFileFind	Pointer to the file search information handler structure. The structure should be allocated before the function call and it will be filled inside the function's operating progress.
disk_letter	The SLFS disk name. May be 'A','B','C' or 'D'

Return parameter:

ERR_NO_ERR	The file was found and *'hFileFind'* structure contains the file information (if SLFS contains at least one file)
ERR_FILE_NOT_FOUND	There are no files in SLFS
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function is used to start file enumeration procedure in the SLFS file system.
If function returns ERR_NO_ERR, the '*hFileFind*' structure contains the first found file information.

slfs_enum_files_next

Continue file enumeration

int slfs_enum_files_next(**SLFS_FILEFIND** * hFileFind)

Parameter:

hFileFind	Pointer to the file search information handler structure.

Return parameter:

ERR_NO_ERR	The file was found and *'hFileFind'* structure contains the file information
ERR_LAST_PAGE	File not found (there are no more files to enumerate)
Other error codes	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

This function is used to continue file enumeration process. If the function returns ERR_NO_ERR, the '*hFileFind'* structure contains the enumerated file information.

To provide the SLFS files enumeration, the function ***slfs_enum_files_first()*** should be invoked first.
Then, if ***slfs_enum_files_first()*** function returns ERR_NO_ERR, the function ***slfs_enum_files_next()*** should be called in the loop while it will return ERR_NO_ERR.
For more details, see an example.

slfs_get_files_number

Get the actual file number

int slfs_get_files_number(**DF_FLI** * fli,
int disk_letter)

Parameter:

fli	pointer to the global flash descriptor structure
disk_letter	The SLFS disk name. May be 'A','B','C' or 'D'

Return parameter:

Greater than or equal to zero	Actual number of files in the SLFS disk
Less than zero	Wrong parameters or internal errors. For the error code value see the file *sys_errorcodes.h*

Description:

The function returns actual number of files in SLFS disk or error code.
It uses ***slfs_enum_files_first() / slfs_enum_files_next()*** functions internally.