

smxFS™

Portable FAT File System

smxFS is a FAT file system that is media-compatible with DOS/Windows. It has small code and data footprints, making it ideal for small embedded systems. smxFS supports flash media such as USB thumb drives, CompactFlash, and SD/MMC. It provides the standard C library file API.

smxFS is a FAT file system for hard real-time embedded systems. It supports fixed and removable media, and offers drivers for the media typically used in modern embedded systems, such as USB thumb drives, CompactFlash, and SD/MMC cards. It is DOS/Windows media-compatible, so that media written by smxFS are interchangeable with Windows and other RTOSs that support the FAT file system. smxFS requires minimal ROM and RAM, allowing it to be used in very small embedded systems, as well as in larger ones. The Lite version reduces memory requirements further. smxFS supports FAT12/16/32 and VFAT (long file names compatible with Win32 operating systems). It uses the standard C library file API (i.e. `open()`, `fread()`, etc.) that is familiar to most C programmers, and it is extended with additional functions for other needed operations. smxFS API functions are reentrant so that smxFS is safe for multitasking.

Components

smxFS has the following components:

1. FS API provides the standard C library API (`open()`, `fread()`, `fwrite()`, `fseek()`, `fclose()`, etc.) to the application.
2. FS Path implements the Directory Entry and FAT table structure handler. It supports FAT12/16/32 and VFAT.
3. FS Mount implements the mount, unmount, and format functions for inserted devices.
4. FS Cache implements the Cache for Data, FAT, and Directory entries.

Features

- FAT 12/16/32
 - Long File Names (VFAT)
 - DOS/Windows Media Compatible
 - Media Drivers:
 - USB Mass Storage
 - SD/MMC (4-bit bus or SPI)
 - DiskOnChip®
 - CompactFlash
 - ATA/IDE Hard Drive
 - RAM Disk
 - NAND Flash Disk
 - NOR Flash Disk
 - Serial NOR Flash Disk
 - Up to 2 Terabyte Disk Size
 - Compatibility with ARM, ColdFire, PowerPC, x86, and other CPUs
 - Supports 16-bit addressing CPUs such as TI TMS320C55xx DSPs
 - 15 KB Minimum Code Footprint
 - 3 KB Minimum Data Footprint
 - Standard C Library file API
 - Subdirectory Support
 - Partitioning Support
 - Multitasking Support
 - `chkdsk` and Clean Shutdown Check
 - Source Code Included
 - Optimized for SMX® RTOS
 - Easily Portable or Runs Standalone
5. FS Driver Interface integrates all of the devices into the file system via a unique interface.

6. **FS Port** implements the OS and Compiler related definitions, macros, and functions.
7. **FS Utility** implements check disk and fix functions.

Driver Interface

Seven driver interface functions have been defined to support all kinds of removable media. By using this interface, smxFs does not need to know the details of the media. It treats all devices according to the same interface. If you want to add a new device driver, just implement these functions and call `sfs_devreg()` to register it with smxFs. No changes to smxFs files are required.

```
int  DriverInit (void)
int  DriverRelease (void)
int  DiskOpen (void)
int  DiskClose (void)
int  SectorRead (addr, startsector, num)
int  SectorWrite (addr, startsector, num)
int  IOctl (command, parameter)
```

Drivers

smxFs offers many drivers. See the smxFs User's Guide, section Driver-Specific Notes, for more information.

ATA

Supports basic operations.

CompactFlash

Supports memory mode.

DiskOnChip®

Based on TrueFFS v6.x and supports G3/P3, G4, and H1 families.

MMC/SD

Supports 4-bit bus and SPI.

NAND

See smxFFS datasheet.

NOR

See smxFD datasheet. Not intended for frequent modification of files or logging. Instead use NAND flash or smxFLog.

RAM Disk

Included.

USB

See smxUSBH datasheet.

Windows

Allows running smxFs under Windows, which may be helpful during development. Included with smxSim and standalone (non-SMX) orders.

Porting Layer

Two files, `fport.h` and `fport.c`, contain definitions, macros, and functions to port smxFs to any OS, compiler, and processor. Currently, they support SMX®, Windows, and Linux.

Configuration

smxFs has a simple configuration file, `fcfg.h`, to allow setting preferences and tuning RAM and ROM requirements. Cache characteristics can be set to control RAM usage. Some options can be disabled to reduce code usage, such as FAT32, VFAT, and write support.

16-Bit Addressing Support

smxFs supports processors that can only do 16-bit memory addressing (not byte addressing) such as the TI TMS320C55xx DSPs. These processors are difficult to support because of byte data and byte fields in standard data structures. This is enabled by a configuration option in smxFs.

Safety

The DOS/Windows FAT file system is inherently not power fail safe, so smxFs implements features to compensate for this:

1. `sfs_chkdisk()` API can check and fix many problems, such as cross-linked files, lost chains, bad directory entries, and other problems. Flags specify which types of problems to fix (or none). It indicates the results using flags in the return value and can give detailed text information in a buffer, to allow a human operator to correct problems. Please see the smxFs User's Guide for details.
2. Clean shutdown checking determines whether all files were closed and caches flushed before the system shut down. If not, `sfs_chkdisk()` should be called.

File System API

+ sfs_init ()	Initialize the File System.
+ sfs_exit ()	Uninitialize the File System.
+ sfs_devreg (dev_if, id)	Register a device driver to the system.
+ sfs_devunreg (id)	Un-Register a device driver from the system.
+ sfs_devstatus (id)	Return the current status of the device/disk (e.g. mounted).
+ sfs_getdev (id)	Return pointer to the device information structure.
+ sfs_fopen (filename, mode)	Open a file for read/write access.
+ sfs_fclose (filehandle)	Close an open file and flush all data to the storage media.
+ sfs_fread (buf, size, items, filehandle)	Read data from an open file.
+ sfs_fwrite (buf, size, items, filehandle)	Write data to an open file.
+ sfs_fseek (filehandle, offset, method)	Move the file pointer to the specified location.
+ sfs_fdelete (filename)	Delete a file.
+ sfs_remove (filename)	Alias for sfs_fdelete ().
sfs_delmany (filelist, num)	Delete many files in one operation.
sfs_copy (src, dest)	Copy a file to the same or different volume.
sfs_move (oldname, newname)	Alias for sfs_rename ().
sfs_rename (oldname, newname)	Rename a file or directory. Can move a file anywhere or a directory to the same volume.
sfs_chmod (filename, mode)	Set a file's permission settings.
sfs_timestamp (filename, datetime)	Set a file's modification timestamp.
sfs_getprop (filename, fileinfo)	Get a file's properties, including attribute, size, and timestamps.
sfs_setprop (filename, fileinfo, flag)	Set a file's attribute and/or timestamps (flag indicates which).
sfs_stat (filename, fileinfo)	Get information about the file.
+ sfs_filelength (filename)	Return the length of a file, in bytes.
+ sfs_rewind (filehandle)	Move the file pointer to the beginning of the file.
+ sfs_fflush (filehandle)	Flush all data associated with the file handle to the storage media.
+ sfs_ftell (filehandle)	Determine the current file pointer position.
sfs_truncate (filehandle)	Truncate a file at the current file pointer.
+ sfs_feof (filehandle)	Test for end-of-file.
+ sfs_findfile (filename)	Test if a file exists.
sfs_findfirst (filespec, fileinfo)	Provide information about the first instance of a file whose name matches the name specified by the <i>filespec</i> argument.
sfs_findnext (id, fileinfo)	Find the next file, if any, whose name matches the <i>filespec</i> argument in a previous call to sfs_findfirst (), and return information about it in the <i>fileinfo</i> structure.
sfs_findclose (fileinfo)	Clean up after the findfirst / findnext operation.
+ sfs_mkdir (path)	Create a new directory.
+ sfs_rmdir (path)	Remove a directory.
sfs_chdir (path)	Alias for sfs_setcwd ().
sfs_getcwd (buf, maxlen)	Get the current working directory for the current task.
sfs_setcwd (path)	Change the current working directory for the current task.
sfs_chkdisk (id, fixflag, buf, len)	Check and fix disk problems. Flags specify what problems to fix.
+ sfs_partition (id, partitioninfo)	Partition the storage media.
+ sfs_format (id, formatinfo)	Format the storage media.
sfs_getvolname (id, name)	Get a disk's volume name.
sfs_setvolname (id, name)	Set a disk's volume name.
+ sfs_freekb (id)	Return the number of free KB on the storage media.
+ sfs_totalkb (id)	Return the number of total KB on the storage media.
+ sfs_writeprotect (id)	Determine if the media is write protected.
+ sfs_ioctl (id, command, par)	Perform device-specific functions.

+ Included in smxFS Lite

Size and Performance

Code and Data Size

Code size varies depending upon CPU, compiler, and optimization level.

RAM is allocated for caching data sectors, FAT table sectors, and directory entries. The cache size depends upon the sector size. If sector size is 512 bytes, then 1.5KB is enough for the system to run. Of course, increasing the cache size will increase performance. (Using a 1.5KB cache instead of the default 22KB reduces performance to 20%.) RAM is also needed for the stack. For a single task (or non-multitasking), about 100 bytes are needed, plus 600 bytes to support long file names (which can be disabled). For the USB disk driver, another 400 bytes are needed.

	ARM7/9 IAR ROM (KB)	ColdFire CodeWarrior ROM (KB)	RAM (KB)
API and core files (max ¹)	27.5	30.0	2.2 ⁶
API and core files (reduced ²)	11.0	12.0	2.2 ⁶
ATA driver	1.0	1.5	—
CompactFlash driver	1.0	1.5	—
MMC/SD driver (bus)	6.0	N/A	0.8
MMC/SD driver (SPI)	N/A	7.0	0.8
NAND flash driver	9.0	10.0	see smxFFS
NOR flash driver ³	6.0	7.0	0.7 + see smxFD
RAM disk driver	0.5	0.5	disk size
USB disk driver ⁴	32.5	35.0	0.4 + see smxUSBH

Notes:

1. Unused API functions are dead-stripped by the linker, so the size used by your application is likely to be much smaller.
2. Reduced size is with SFS_READONLY enabled and SFS_FULL_FEATURES_SUPPORT disabled.
3. The NOR flash driver code is part of smxFD. It includes two layers, STL and HIL. HIL is used to port to different hardware and flash chips so the size can vary widely. The above size is for the Intel 28F128K3 StrataFlash NOR flash chip.
4. The USB disk driver code is part of smxUSBH. The size can vary widely depending upon which Host Controller driver you are using. The size above includes the ISP1362 driver. See the smxUSBH brochure for other sizes.
5. Driver sizes vary because they have porting code that varies for different hardware.
6. Minimum cache + stack sizes. See discussion above table.

Performance

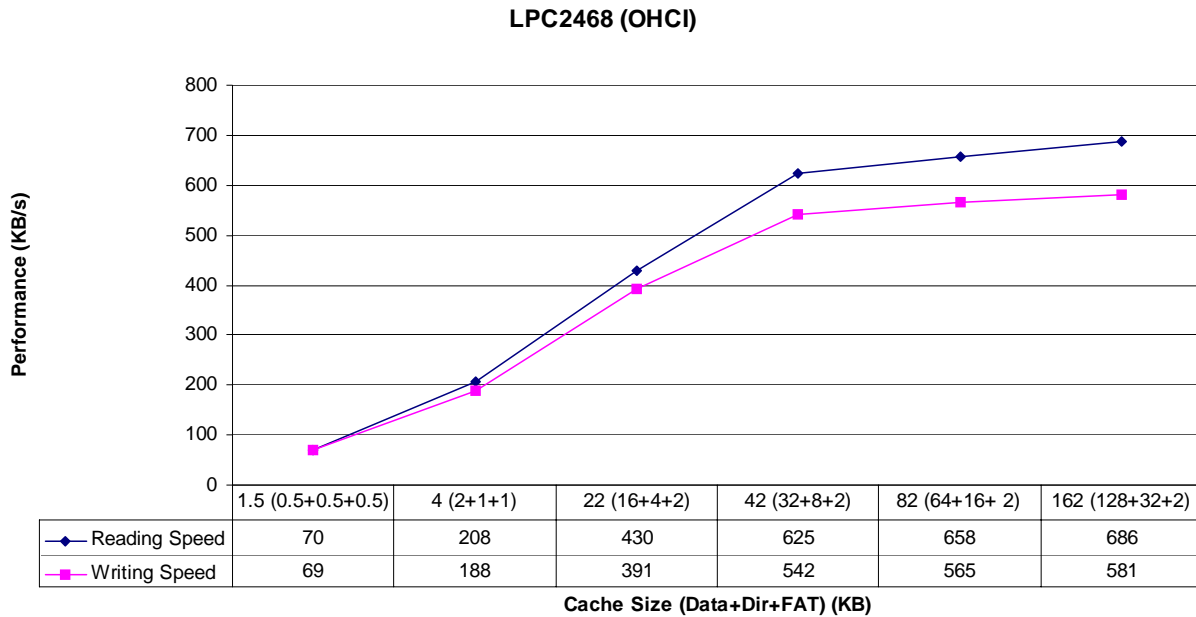
Media	Reading (KB/s)	Writing (KB/s)
CompactFlash driver	919	695
MMC/SD driver (bus mode, LPC2468) ¹	970	870
MMC/SD driver (SPI, Avnet 5282 board) ²	399	454
MMC/SD driver (SPI, STR912 KickStart board) ³	920	1150
MMC/SD driver (SPI, STR912 KickStart board) ⁴	920	1210
NOR flash driver ⁵	950	110
USB driver, flash disk (EHCI) ⁶	10556	7787
USB driver, flash disk (OHCI)	885	817
USB driver, flash disk (UHCI)	611	590
USB driver, flash disk (ISP116x)	336	328
USB driver, flash disk (ISP1362)	591	476
USB driver, flash disk (ISP176x)	7023	3072
USB driver, flash disk (AT91SAM9260)	555	505
USB driver, flash disk (AT91SAM9261)	458	414
USB driver, flash disk (EP9315)	575	498
USB driver, hard disk (EHCI)	20078	16786

Notes:

1. MMC/SD bus mode times were measured using DMA.
2. MMC/SD SPI times were measured using polling, which is the best case.
3. Measured using Sandisk 256 MB, Ultra II 512 MB, and 1 GB SD cards.
4. Measured using Sandisk Extreme III 2 GB SD card.
5. NOR flash times are for Intel 28F128K3 StrataFlash on MCF5485EVB.
6. USB times vary significantly depending on the controller, as the examples show. See the Performance section of the smxFS User's Guide for details. Also see the smxUSBH brochure.

Cache Sizes vs. Performance

The size of caches affects the performance. The following graph shows the performance of a USB flash disk reading/writing operation when the cache sizes are set to different values. See the smxFS User's Guide for more graphs for different hardware.



Features Omitted from the Lite Version

smxFS Lite omits advanced features to reduce memory footprint and save cost. It is a subset of the same codebase as the full version. The Lite version has the following limitations:

- Long File Name support is omitted. You must use 8.3 format names to access files and paths.
- FAT32 is disabled by default but you can set `SFS_FAT32_SUPPORT` to 1 to enable it again.
- Current Working Directory is disabled. You must use full path names such as `A:\dir\subdir\file1` to access files. Related APIs are omitted from the source code.
- The following APIs are omitted:
 - `chkdsk()`
 - `findfirst()/findnext()/findclose()`
 - `rename()`
 - `setprop()/getprop()/stat()/chmod()/timestamp()`
 - `setvolname()/getvolname()`
- The following APIs are disabled by default:
 - `mkdir()/rmdir()`. Can enable by setting `SFS_MKDIR_SUPPORT` to 1.
 - `format()/partition()`. Can enable by setting `SFS_FORMAT_SUPPORT` to 1. These functions are provided primarily to support fixed media, since removable media are preformatted and can be reformatted on a PC.