

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 OSs 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. 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. fopen(), 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 (fopen(), 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.
5. FS Driver Interface integrates all of the devices into the file system via a unique interface.
6. FS Utility implements check disk and fix functions.

*VFAT is patented and may require a license from Microsoft. Contact lplicreq@microsoft.com.

Features

- FAT 12/16/32
- Up to 2 Terabyte Disk Size
- Long File Names (VFAT), including international characters such as Chinese
- DOS/Windows Media Compatible
- Media Drivers:
 - USB Mass Storage
 - SDHC/SD/MMC (4/8-bit bus or SPI)
 - CompactFlash
 - NAND Flash Disk
 - NOR Flash Disk
 - Serial NOR Flash Disk
 - DiskOnChip®
 - ATA/IDE Hard Drive
 - RAM Disk
- Compatibility with ARM, ColdFire, PowerPC, x86, and other CPUs
- Supports non-byte addressable CPUs such as TI TMS320C55xx DSPs
- 15 KB Minimum Code Footprint
- 3 KB Minimum Data Footprint
- Full performance and memory usage information on real hardware.
- Standard C Library file API
- Subdirectory Support
- Multiple Partition/Disk Support
- Multitasking Support
- chkdsk and Clean Shutdown Check
- Source Code Included
- Optimized for SMX® RTOS
- Easily Portable or Runs Standalone
- PC Emulator

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. These drivers are designed to be portable to different hardware. Contact us if you don't see your specific hardware supported.

ATA

Supports basic ATA operations with polling. No interrupt nor DMA.

CompactFlash

Supports memory mode. A simple API is provided to port it to your hardware.

DiskOnChip®

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

MMC/SD

Supports 4/8-bit bus and SPI. The driver has been designed to separate the higher level code (MMC/SD protocol) from the hardware (controller) interface code. There is no standard for SPI or bus mode controllers but we have tried to define a general low level API that can support them all. Has low-level drivers to interface to the SD/MMC controllers on various SoCs. Streaming mode (multiple block) mode is supported. The driver automatically detects whether an inserted card is MMC or SD during the Identification phase.

NAND

See smxNAND datasheet.

NOR

See smxNOR 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

The porting layer is mostly contained in smxBase. Two files, `fport.h` and `fport.c`, contain additional definitions, macros, and functions. SMX® and several other RTOSes are supported.

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.

Non-Byte 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.

the system shut down. If not, `sfs_chkdisk()` should be called.

2. Clean shutdown checking determines whether all files were closed and caches flushed before

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 <code>sfs_fdelete()</code> .
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 <code>sfs_rename()</code> .
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 <code>sfs_findfirst()</code> , and return information about it in the <i>fileinfo</i> structure.
sfs_findclose (fileinfo)	Clean up after the <code>findfirst/findnext</code> operation.
+ sfs_mkdir (path)	Create a new directory.
+ sfs_rmdir (path)	Remove a directory.
sfs_chdir (path)	Alias for <code>sfs_setcwd()</code> .
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 reduced configuration. See size table below.

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 caches. Of course, increasing the cache size will increase performance. (Using a 1.5KB cache instead of the default 22KB reduces performance by 80%.) 0.5KB is allocated for the long file name UNICODE buffer. Another 0.5KB is used for disk handler information and other global variables such as current working directory for each task. RAM is also needed for the stack. For a single task (or non-multitasking), about 600 bytes are needed. For the USB disk driver, another 400 bytes are needed.

	ARM7/9 IAR ROM (KB)	ColdFire CodeWarrior ROM (KB)	Minimum RAM (KB)
API and core files (max ¹)	42.0	50.0	4.0 ⁶
API and core files (reduced ²)	17.0	23.0	4.0 ⁶
ATA driver	1.0	1.5	—
CompactFlash driver	1.0	1.5	—
MMC/SD/SDHC driver (bus)	6.5	N/A	0.8
MMC/SD/SDHC driver (SPI)	N/A	7.5	0.8
NAND flash driver	9.0	10.0	see smxNAND
NOR flash driver ³	6.0	7.0	1.2 ³
RAM disk driver	0.5	0.5	disk size
USB disk driver ⁴	28.0	34.0	0.4 + see smxUSBH

Notes:

1. Max is the size with nearly everything enabled. 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_FULL_FEATURES_SUPPORT disabled. Various features can be enabled independently. See the smxFS User's Guide for sizes of these options.
3. The smxNOR flash driver code 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. The RAM size includes a 512-byte sector buffer for readback verify, which may not be needed in your case.
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 + one sector buffer + global variable + stack sizes. Size is for one open file. See discussion above table.

Performance⁷

Media	Reading (KB/s)	Writing (KB/s)
CompactFlash driver	1258	1090
MMC/SD/SDHC driver (bus mode, SAM9RL64) ¹	5389	4471
MMC/SD/SDHC driver (SPI, STR912 KickStart board) ²	1070	1017
MMC/SD/SDHC driver (SPI, STR912 KickStart board) ³	1192	1023
NAND flash driver ⁴	2909	1652
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 and SanDisk Extreme III 2GB SD card.
2. Measured using Sandisk Extreme III 2 GB SD cards.
3. Measured using Sandisk 8 GB SDHC card.
4. NAND flash times are for Samsung K9F2G08U0M on AT91SAM9263-EK.
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.
7. Performance is tested by writing 4KB data at a time up to 20MB. Increasing the write buffer size and the cache size can increase performance up to 80%. All the testing is based on the default cache setting which is 22KB.

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.

LPC2468EA OEM Board 72MHz (USB OHCI):

