# TFS4
# INITIALIZATION PROCESS

## Application Note

November-2006, Version 1.0

SAMSUNG
ELECTRONICS

# Copyright Notice

**Copyright ® 2006, Flash Software Group, Samsung Electronics Co., Ltd**

**All rights reserved.**

**Trademarks**

TFS4 is a trademark of Memory Division, Samsung Electronics Co., Ltd in Korea and other countries.

## Restrictions on Use and Transfer

## Contact Information

Flash Software Group
Memory Division
Samsung Electronics Co., Ltd

**Address:** San #16 Banwol-Dong, Taean-Eup
Hwasung-City,Gyeonggi-Do, Korea, 445-701

# Introduction

Along with more use of embedded system, the data storage and management in it has become an important research problem. Flash memory has many advantages as high speed and low cost and so on, thus to be used more and more in embedded systems. In the meantime, in order to manage the storied data reasonably and share data, as embedded RTOS, embedded file system has become a trend. Generally, the filesystems can't be directly used in embedded systems. Currently, there are many mature file systems on computer, such as FAT, NTFS and so on. But unlike such disc or compact disc storage medium on general PC, there are RAM and ROM and Flash in embedded systems only with low capacity. So generally, these file systems can't be directly used in embedded systems.TFS4 is a flash file system offered by Samsung which integrates FTL and FAT file system added externally to RTOS. TFS4 has basic functionality as traditional file system that organizes directories and files in storage devices like hard disk drive or flash memory. Additionally, TFS4 has other features for managing data on a specific storage device, NAND flash memory, and MMC (Multimedia Card).TFS4 is composed of several components.

- ■File system abstraction layer
- ■KFAT filesystem
- ■Buffer Cache Manager
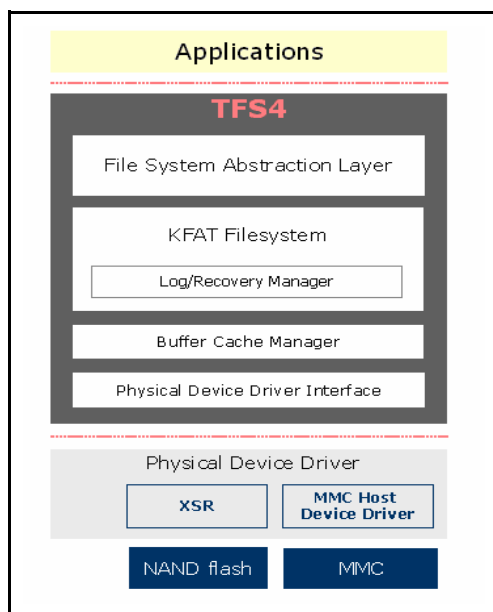- ■Physical Block Device Driver



**Figure 1. TFS4 System Architecture**

TFS4 can use several physical devices simultaneously and is compatible with FAT filesystem. The existing FAT filesystem has weakness of power off recovery. However, TFS4 is well designed in consideration of it, and supports fast recovery when the power is suddenly off. TFS4 guarantees the integrity of meta-data of FAT filesystem. For higher portability and easier maintenance, TFS4 has the layered architecture.

The first step in order to use the file system is to register with the OS. Initialization makes the file system ready. This process is called Mount operation.

During booting, the application wants to access the file system. At that time the file system should be mounted/initialized.
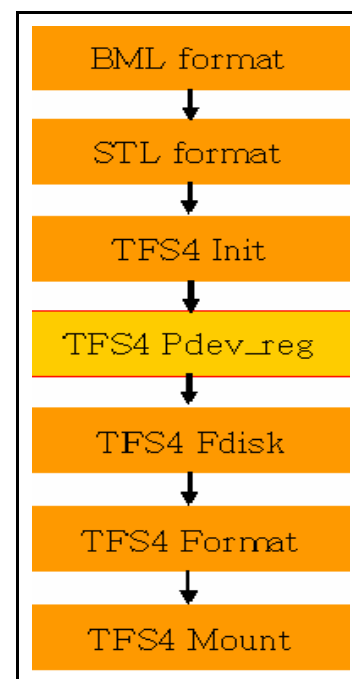
# Steps to Initialze



**Figure 2. TFS4 Initialization Procedure**

There are 2 main steps: First is FTL-XSR initialization and second is file system initilization.

XSR initialization consists of 2 steps:

1) BML Format and 2) STL format.

## BML Format

Tfs4_pdev_nand_xsr.c-->
tfs4_pdev_nand_xsr_bml_format();
 r = tfs4_pdev_nand_xsr_bml_format();

## STL Format

Tfs4_pdev_nand_xsr.c-->
tfs4_pdev_nand_xsr_stl_format();
 r = tfs4_pdev_nand_xsr_stl_format();

We can use the block device driver after  initializing XSR. Later, we must initialize file system.

## TFS4 Init

Initializes TFS4 system shell by the command tfs4_init.

## Syntax

t_int32 tfs4_init(void);

## Return value

0 - TFS4 File system intialized successfuly.
Under - 1 Initialization failure

This function initializes TFS4 file system. To use TFS4 file system, this function should be called just once, at first. Generally, it is called once when an OS is booted. Thus, when some application programs are made, it should not be called again. When this function returns a negative value, TFS4 file system cannot be used any more.

Tfs4_volume_manager.c --> Tfs4_init();
 r = tfs4_init();

## Physical device Registration

Register devices by using the command pdev_reg

## Syntax

t_int32 tfs4_pdev_reg(const  t_physical_device_op *pOp,t_boolean bInit, t_boolean bOpen)

## Parameters

■*pOp - physical device I/O operation function pointer structure.

■bInit - Initialize physical device while registration.
■bOpen- Open physical device while registration.

## Return value

0 - TFS4 File system intialized successfuly.
Under - 1 Initialization failure

Tfs4_pdev_nand_xsr.c-->
tfs4_pdev_nand_xsr_get_op(), tfs4_pdev_reg
t_physical_device_opstOp;

TFS4_memset(&stOp,0x00,TFS4_sizeof(t_physical_device_op));

r = tfs4_pdev_nand_xsr_get_op(&stOp);
r = tfs4_pdev_reg(&stOp, TRUE, TRUE);

File System Fdisk - Making Partition

Tfs4_api.c  -->  tfs4_ioctl(),  tfs4_fdisk_delete(), tfs4_fdisk_new()

t_physical_device_infostPDevInfo;
t_pinfostPI;
t_int32 r;

r=tfs4_ioctl("/dev/nf",enuIOCTL_PDEV_STAT, &stPDevInfo);

r=tfs4_fdisk_delete(stPDevInfo.szDeviceName,0); /
/nf0: 0 , nf1: 1 ....

TFS4_memset(&stPI, 0x00, TFS4_sizeof(t_pinfo));

stPI.dwPN = 0;  // partition number, /dev/nf0 ==> index 0
stPI.bBootDescriptor = 0x80;    // Bootable
stPI.dwStartSector = 1;  // Start sector is 1
stPI.dwEndSector  =  stPDevInfo.dwEndSector;  // end sector is last sector
stPI.bPartitionType = 0x04;// no Extended Partition
stPI.dwNumSectors     =     stPI.dwEndSector    - stPI.dwStartSector + 1;
r=tfs4_fdisk_new(stPDevInfo.szDeviceName, &stPI);

## Format drive

The function formats the logical partition that psDeviceName specifies to a psFilesystemType.

psFilesystem specifies FAT types. It is chosen among    TFS4_FAT12,    TFS4_FAT16,    and TFS4_FAT32.  dwClusterSize is sector count per a cluster (the size of one sector is 512-Byte.)

psDeviceName is one of /dev/nf[0-3] or /dev/mmc0.

psFilesystemType uses TFS4_KFAT_FS constants.

dwClusterSize should be a square of 2, from 1 to 64.

t_int32 tfs4_format(const t_char *psDeviceName, const t_char *psFilesystemType, const t_char *psFilesystem, t_uint32 dwClusterSize);

## Parameter

■psDeviceName Device name to be formatted
■psFilesystemType Filesystem format type
■psFilesystem FAT type
■dwClusterSize Cluster size in sector count.

## Return value

0 - Success
-1 Failure

Tfs4_api.c --> tfs4_format()
 r = tfs4_format("/dev/nf0", "KFATFS", "FAT16", 8);

## Mount drive

This function mounts NAND or MMC partition onto a file system volume.

tfs4_mount() attaches the filesystem specified by psDevice to the volume specified by psTarget.

t_int32 tfs4_mount ( const t_char *psDevice, const t_char *psTarget,const t_char *psFilesystemType, t_uint32 dwFlag, const void *pData);

## Parameters

■*psDevice The logical device name. Logical device name consists of physical device and partition number.
■*psTarget-Target volume name.
■*psFilesystemType - This is filesystem name. Current version of TFS4 provides "KFATFS" only."KFATFS" is fully FAT compatible filesystem.
■dwFlag - Mount flag.

Flags for TFS4 hidden area

TFS4_MOUNT_HDIR,
TFS4_MOUNT_HDIR_CREATE
TFS4_MOUNT_HDIR_SHOW

These flags are available when TFS4 is built with TFS4_HIDDEN_DIR module.

TFS4_MOUNT_HDIR : It mounts a volume with hidden area.

tfs4_mount() will fail with this flag when there is no hidden area on the volume.

TFS4_MOUNT_HDIR_CREATE: TFS4 creates hidden area when there is no hidden area. It enables hidden area when there is a hidden area without hidden area creation.

TFS4_MOUNT_HDIR_SHOW: It makes the hidden directory to a normal directory. This hidden directory will be shown at root directory of mounted volume.

Tfs4_api.c --> tfs4_mount()
r = tfs4_mount("/dev/nf0", "/a/", "KFATFS", 0, NULL);

■pDATA In current version of TFS4 does not use this parameter. This is for POSIX compatibility. Just set this parameter as NULL

## Return value

0 - Success
-1 Failure

# References

**1.Samsung Electronics TFS4 Programer's Guide**
**2.Samsung Electronics XSR1.5 Porting Guide**