TFS4 v1.5.0

Porting Guide

2006.05.02 , Version 1.5.0

Note

TFS4 is independent of XSR. Here we assume that XSR or MMC (or HSMMC) host device driver is already ported to your target system. This TFS4 porting guide covers only TFS4 porting procedure, neither XSR nor MMC (or HSMMC) host device driver.

SAMSUNG ELECTRONICS

Copyright notice

Trademarks

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

Restrictions on Use and Transfer

All software and documents of **TFS4** are commercial software.
Therefore, you must install, use, redistribute, modify and purchase only in accordance with the terms of the license agreement you entered into with Memory Division, Samsung Electronics Co., Ltd.

All advertising materials mentioning features or use of this software must display the following acknowledgement:
This product includes **TFS4** written by Memory Division, Samsung Electronics Co., Ltd.

Contact Information

Flash Software Group
Memory Division,
Samsung Electronics Co., Ltd

Address: San #16, Banwol-Dong, Hwasung-City, Gyeonggi-Do, Korea, 135-120

Preface

SEC-MEM-TFS4-POG00001

This document is a porting guide of TFS4 developed by Embedded Storage team, Memory Division, Samsung Electronics. It describes TFS4 porting procedure to user's target platform and OS.

Purpose

This document helps engineers of Samsung Electronics or other companies, who use TFS4 as an embedded file system. You can understand the way TFS4 is ported in your target system, and follow the porting procedure easily with this document.

Scope

This document covers an overview, system requirements, source directory structure, and the detailed porting procedure of TFS4. It also describes TFS4 test case to find if TFS4 is correctly running after porting.

Definitions and Acronyms

Definitions and Acronyms	Description
BPB	Bios Parameter Block
Block	Erase unit of the NAND flash memory
BML	Block Management Layer
Cluster	Read/write unit of the filesystem
EOC	End Of Cluster
FAL	File system Abstraction Layer
FAT	File Allocation Table
File System	General file system that provides total file system service
Filesystem	Specific filesystem such as FAT, EXT2, and NTFS
FTL (Flash Translation Layer)	A software module which maps logical addresses to physical addresses when accessing flash memory
Initial bad block	Invalid blocks on arrival from the manufacturers
Interleaving	Parallel data I/O from different devices
KFAT	A new type of filesystem that is used for TFS4
LBA	Logical Block Addressing
LLD	Low Level Device Driver
MMC	Multimedia Card
HSMMC	High Speed Multimedia Card

NAND flash controller	Controller for the NAND flash memory
NAND flash device	Device that contains NAND flash memory or NAND flash controller.
NAND flash memory	NAND-type flash memory
Page	Read/write unit of the NAND flash memory
Sector	Unit of the physical I/O
STL	Sector Translation Layer
TFS	Transactional File System
XSR	eXtended Sector Remapper
MBCS	Multi Byte Character Set, (SBCS + DBCS)
DBCS	Double Byte Character Set
SBCS	Single Byte Character Set

Related Documents

- Microsoft Extensible Firmware Initiative FAT32 File System Specification, Microsoft Corporation, Version 1.03, December 6, 2000
- Long Filename Specification, Microsoft Corporation, Version 0.5, December 4, 1992
- SEC Memory Division, XSR v1.5.0 Part 1. Sector Translation Layer Programmer's Guide, Samsung Electronics, Co., LTD, DEC-09-2004
SEC Memory Division, XSR v1.5.0 Part 2. Block Management Layer Programmer's Guide, Samsung Electronics, Co., LTD, DEC-09-2004
- SEC Memory Division, XSR v1.5.0 Porting Guide, Samsung Electronics, Co., LTD, DEC-09-2004

History

Version	Date	Comment	Author	Approve
0.1	2005.12.06	Pre-released version	Embedded Storage System	
0.2	2004.04.02	Reviewed by OS Solution Lab.	Sun Mee Kwak	
1.0	2004.04.12	1.0 Release version	Sun Mee Kwak	Kyung il Bang
1.2.0	2004-07-28	For TFS4 1.2	DongYoung Seo	
1.2.0_patch001	2004-11-05	Modify memory usage for TFS4 1.2_patch001	DongYoung Seo	
1.3.0	2005-02-24	Re-write for TFS4 v1.3	DongYoung Seo	
1.4.0	2005-03-28	Initial document for v1.4	DongYoung Seo	
1.4.0a	2005-08-12	Modify for XSR v1.4.0	DongYoung Seo	
v1.4.1_build001	2005-10-12	Add Fast Unlink and Fast Seek description	DongYoung Seo	
v1.5.0_build001	2005-11-09	Initial document for v1.5.0	DongYoung Seo	
v1.5.0	2006-01-23	Review	MoonSang Kwon	

Contents

Figures

Tables

1. TFS4 Introduction

TFS4 (Transactional File System 4) is an embedded file system to use the NAND flash memory in the most stable and effective way. This chapter introduces TFS4 briefly. The TFS4 overview, software architecture, and features are explained in this chapter.

1.1. TFS4 Overview

TFS4 is an embedded flash file system to use NAND flash memory as storage on any consumer electronic devices. It provides file system services to application and operating system.

TFS4 overcomes the existing FAT weakness over the power off recovery, and is fast recovered even if the power is suddenly lost. It is fully compatible with FAT file system that has been used in most operating systems and so multimedia data stored in NAND flash memory can be detected by any other systems.

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, MMC (Multimedia Card) and HSMMC (High Speed Multimedia Card).

TFS4 is composed of several components. The following lists the TFS4 components.

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

TFS4 can use NAND and MMC (or HSMMC) simultaneously and is compatible with FAT filesystem. The existing FAT filesystem has weakness of power off recovery. But 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.

1.2. TFS4 System Architecture

As mentioned above, TFS4 has several components. TFS4 is deployed between applications and hardware such as NAND flash memory or MMC (or HSMMC) host drive. It works in conjunction with an existing operating system or in some embedded applications as the operating system.

Figure 1 shows the system architecture of TFS4.

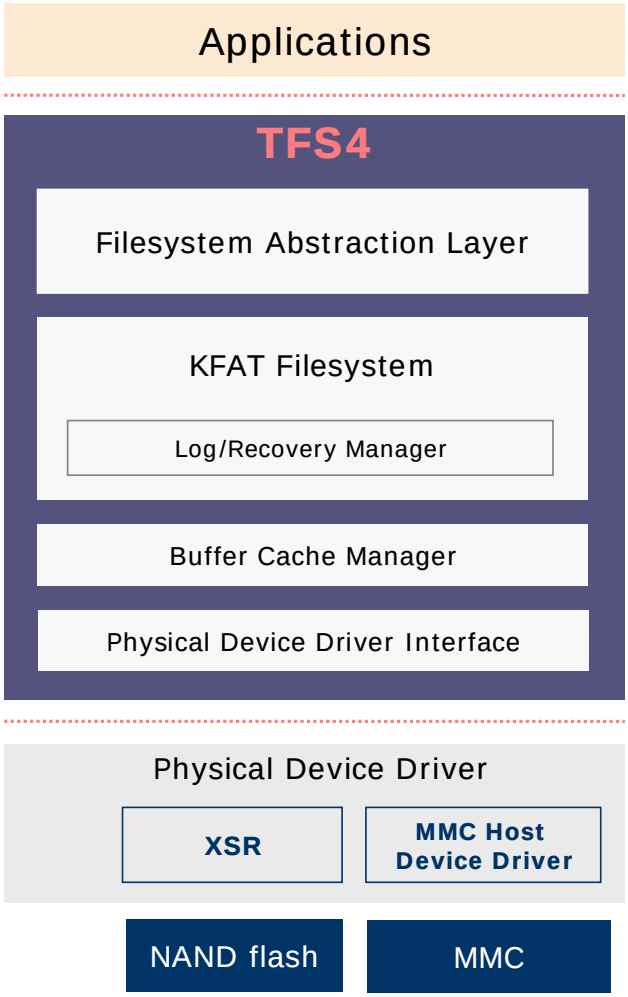


Figure 1-1. TFS4 System Architecture

There are three parts in the above figure. The first is an application at the top of the figure. The second is TFS4 in the middle. The third is a physical device driver and hardware at the bottom of the figure.

TFS4 consists of four modules: File system abstraction layer, KFAT Filesystem, Buffer Cache Manager, and Physical Device Driver Interface.

Application requests filesystem services to file system abstraction layer by using TFS4 APIs. File system abstraction layer receives the requests from application and forwards them to the underlying KFAT filesystem. Then, KFAT filesystem handles the requests with real filesystem operation.

Block device drivers abstract the underlying hardware storage. The storage may be NAND flash and/or MMC (or HSMMC). To use those devices, some kinds of device drivers are needed between the block device driver and storage. The device drivers are XSR and MMC (or HSMMC) host device driver; XSR is a kind of FTL (Flash Transaction Layer) commonly used to manage data on NAND flash.

The following describes the features of each TFS4 component.

- File system Abstraction Layer

File system Abstraction Layer (FAL) is the entry point to the TFS4 File system. It handles all system calls related to the filesystem. It provides a common interface to several kinds of filesystems. Applications may use system calls or other kind of IPC (inter process communication) methods if the TFS4 system operates as a single task to communicate with the file system abstraction layer

- KFAT Filesystem

KFAT provides the real filesystem operations. TFS4 supports FAT16/32 compatible filesystem.

- Buffer Cache Manager

It is a block buffer cache manager for fast I/O with a block device. It’s laid on main memory.

- Physical Device Driver Interface

Physical device driver interface provides TFS4 with an interface to use a physical device driver; it can be a XSR or MMC (or HSMMC) host device driver. The physical device driver interface should be changed to the physical device driver.

To use TFS4, your target system should use NAND flash as a storage device. MMC (or HSMMC) also can be used, but it is optional.

1.3. TFS4 Features

The following describes the main benefits and features of TFS4.

- It supports FAT16/32 filesystem that are commonly used in many systems.
- It supports a long file name. File name can be up to 255 characters and directory name can be up to 243 characters (in Unicode).
- It performs fast power-off recovery.
- It guarantees meta-data integrity of filesystem; meta-data consists of FAT information and directory information.
- It supports both internal and external storage devices.
- The internal storage device can have up to four partitions.
- It can make up to four partitions on the external storage device. And it recognize up to twenty partitions of the external storage device.
- It supports multiple volumes. The default number of volumes is eight and it is configurable up to twenty six.
- It runs stably when the external storage device is suddenly inserted or ejected.
- It supports only an absolute path. Current version does not support a relative path.
- New filesystem features can be added easily, because TFS4 is designed to have virtual file system architecture.

2. System Requirements

This chapter explains the host/target system environment for porting TFS4 to your target system. Host is a development environment. You build the TFS4 image at the host.
Target can be any kind of consumer device using NAND flash memory and MMC (or HSMMC). MMC (or HSMMC) device may not be used for your target, but NAND should be used for TFS4.

2.1. Host

The following table shows the host system requirements for configuring and building TFS4.

Table 1. Host System Requirements

Host Machine	PC
Host OS	Any OS is available.
IDE & Compiler	Any ANSI C/C++ compiler is available. - ADS v1.2 is used to show TFS4 porting in this document.
Source Disk Space	About 5 MB

2.2. Target

TFS4 can be ported to any target, which uses NAND/OneNAND flash memory or MMC (or HSMMC) as a storage. TFS4 has an OS dependent module, which should be ported by the target system designer.

This section specifies the target environment to which TFS4 can be ported. The following table shows the target system requirements.

Table 2. Target System Requirements

Target	Any kind of target device is available. - ReindeerPlus is used in this document as a sample target.
RTOS	Any kind of RTOS is available. - Nucleus is used in this document as a sample OS.
Memory	-Heap: About 230KB. This value is changeable depending on configuration and NAND size -Stack: Maximum 5 KB
Binary Size	About 250KB (Code + Data)

To show a porting example, we describe the porting details about the procedures that the system designer should follow when he ports TFS4 to the ‘ReindeerPlus’ with Nucleus RTOS. ReindeerPlus is a Samsung’s proprietary embedded system development platform which is compatible with S3C2410 board.

This is the hardware information of sample target, ReindeerPlus.

CPU	ARM920T core
Memory	SDRAM: 64M-byte (32M-byte x 2)
UART	Three-channel UART (including IrDA)
MMC	SD host (MMC) interface
Interrupt	EINT interface for MMC In/Out
NAND	It is variable to your target. For ReindeerPlus, A few types of NAND can be used together. The NAND used for test is KFG1G16Q2M (OneNAND.)

Nucleus RTOS is from Accelerated Technology. For fore information, refer the homepage of Accelerated Technology (http://www.acceleratedtechnology.com).

Embedded system has the limited resources. Thus, when you port TFS4 to your target, you have to figure out the TFS4 memory usage.

< TFS4 Static Memory Usage >
TFS4 uses memory that is allocated when TFS4 library is linked. The following lists the TFS4 static memory usage.

```
Buffer cache      = TFS4_CACHE_COUNT * TFS4_SECTOR_SIZE
Read ahead cache= TFS4_BCACHE_READ_AHEAD_COUNT*TFS4_SECTOR_SIZE
FAT cache         = TFS4_FS_FAT_CACHE_SIZE * TFS4_SECTOR_SIZE
Path cache        = TFS4_PATH_CACHE_COUNT * 756
File table        = TFS4_FILE_MAX * 728
Directory table   = TFS4_MAX_DIR_OPEN * 520
File open table   = TFS4_FILE_OPEN_MAX * 28
Volume table      = TFS4_VOLUME_COUNT * 2,320
Etc               = 130,510    (for code page and global variables etc.)
```

Total static memory usage (Byte) = [Buffer cache] + [Read ahead cache] + [FAT cache] + [Path Cache] + [File Table] + [Directory table] + [File open table] + [Volume Table] + [Etc]

If you specify the configuration in the tfs4_config.h or tfs4_config_const.h file as follows, total amount of TFS4 static memory usage is about 238Kbytes.

```
#define TFS4_CACHE_COUNT                    128
#define TFS4_BCACHE_READ_AHEAD_COUNT        8
#define TFS4_SECTOR_SIZE                    512
#define TFS4_FS_FAT_CACHE_SIZE              32
#define TFS4_PATH_CACHE_COUNT               16
#define TFS4_FILE_MAX                       32
#define TFS4_MAX_DIR_OPEN                   16
#define TFS4_FILE_OPEN_MAX                  48
#define TFS4_VOLUME_COUNT                   8
```

TFS4 memory usage can be different according to your configuration.

< TFS4 Dynamic Memory usage>
TFS4 uses dynamic memory for tfs4_chkdsk() only.
You can get the amount of memory for tfs4_chkdsk() as following equation.

Total dynamic memory usage
= [FAT bitmap size] + TFS4_PATH_COMPONENT_MAX *
(TFS4_PATH_NAME_MAX_LENGTH + 724+ 4 +712 + 512 + 1028)

FAT bitmap size = ([Total Cluster count] + 2) / 8 +1
Ex) 128MB NAND Flash, Format FAT16, cluster size 4
[Total Cluster count] = 128 * 1024 *1024 / 512 / 4= 65536
FAT bitmap size = (65536 + 2) /8 + 1 = 8193 byte

If you specify the configuration in the tfs4_config.h or tfs4_config_const.h file as follows, total amount of TFS4 dynamic memory usage is about 80Kbytes.

```
#define TFS4_PATH_NAME_MAX_LENGTH          1024
#define TFS4_PATH_COMPONENT_MAX            16
```

< TFS4 stack usage>
For RTOS, local variables or parameters of a task can be shared by another task and they can be changed. For that reason, each task needs its own task memory region, called a critical section.

But managing the critical section needs a lot of resources. Thus, they should be stored in the task's own memory region that is a stack.

TFS4 uses 5 KB stack per a task except tfs4_chkdsk() API. Tfs4_chkdsk() API uses 10KB stack per a task.

3. Description of Source Files

This chapter describes source code tree of TFS4. You need to find where they are on the host and know what source files should be configured before porting TFS4.

3.1. TFS4 Directory Structure and Descriptions

TFS4 is mostly released as source code. Now let's assume that the TFS4 source package is installed on "C:\TFS4" directory of your host PC.

The following figure shows the installed TFS4 source code tree on your host.

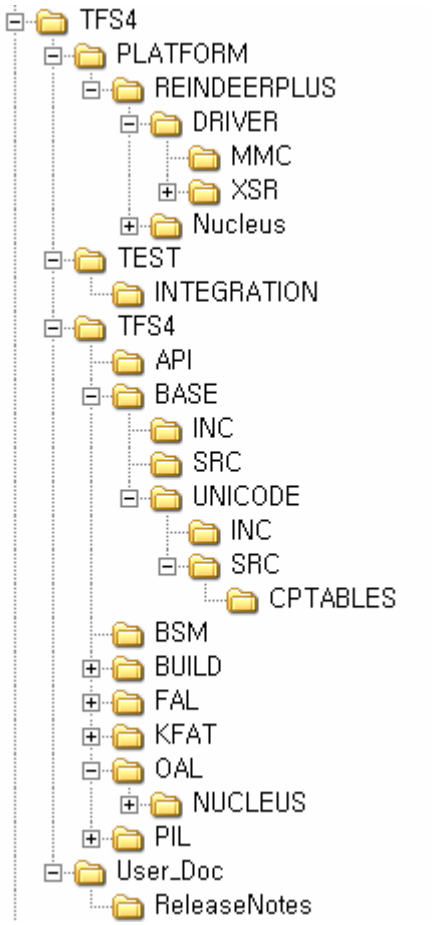


Figure 3-1. TFS4 Directory Structure

The following table describes the TFS4 directories shown in the above picture.

Table 3. Description of TFS4 Directory Structure

Directory	Contains
TFS4	Top directory of TFS4.
TFS4\PLATFORM\REINDEERPLUS	Target OS Source Directory - It is needed when OS dependent module of TFS4 refers to a specific header of target OS. In case of Nucleus, a ***nucleus.h*** file is necessary. - It also includes a device driver for a target. It includes XSR and MMC(or HSMMC) Host Device Driver. For details about XSR, refer to XSR Porting Guide.
TFS4\TEST\INTEGRATION	It has a shell source code for TFS4 testing.
TFS4\TFS4\API	TFS4 API source code
TFS4\TFS4\BASE	It includes base codes for TFS4. The common functionalities are here.
TFS4\TFS4\BSM	It includes Bad Sector Management code.
TFS4\TFS4\BASE\UNICODE	It includes Unicode <-> MBCS conversion source codes and tables
TFS4\TFS4\BUILD\	- It includes a project file to build TFS4 and OS in ADS v1.2. - There is a TFS4 Library Build Project file. - It provides a sample project file to port TFS4 to Nucleus.
TFS4\TFS4\FAL, TFS4\TFS4\KFAT	TFS4 Filesystem Abstraction Layer and KFAT FAT filesystem source code
TFS4\TFS4\OAL	It has OS adaptation source codes. Such as memory, semaphore, terminal and time. You must make the OAL routines for your target and include to the TFS4 library.
TFS4\TFS4\PIL	It has physical device interface source codes for XSR and MMC (or HSMMC)

You can find the project files at “TFS4\TFS4\BUILD\REINDEER_PLUS_ADS12\” and “TFS4\TFS4\BUILD\ADS1_2\” folders. Those project files are created on ADS v1.2. You can use the project files to build TFS4 easily if your build tool is ADS v1.2.

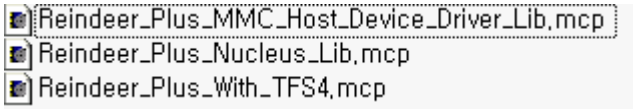


Figure 3-2. TFS4 Project Files at REINDEER_PLUS_ADS12

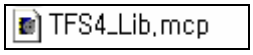


Figure 3-3. TFS4 Project Files at ADS1_2

The following table describes the directories and project files shown in the above picture.

Table 4. Description of TFS4 Project Files

Directory or Project File	Contains
Reindeer_Plus_MMC(or HSMMC)_Host_Device_Driver_Lib.mcp	It includes the source and header files of MMC(or HSMMC) host device driver based on ReindeerPlus.
Reindeer_Plus_Nucleus_Lib.mcp	It includes the Nucleus OS sources that are ported to ReindeerPlus.
TFS4_Lib.mcp	It includes the source and header files that are independent of tfs4_config.h. You can build this project file for making TFS4 library.
Reindeer_Plus_With_TFS4.mcp	It includes the libraries and sources of TFS4, XSR, MMC(or HSMMC) host device driver, and Nucleus OS.

3.2. XSR Directory Structure and Descriptions

This document assumes that they are installed in the "C:\TFS4\PLATFORM\REINDEERPLUS\DRIVER\XSR" directory on your host.

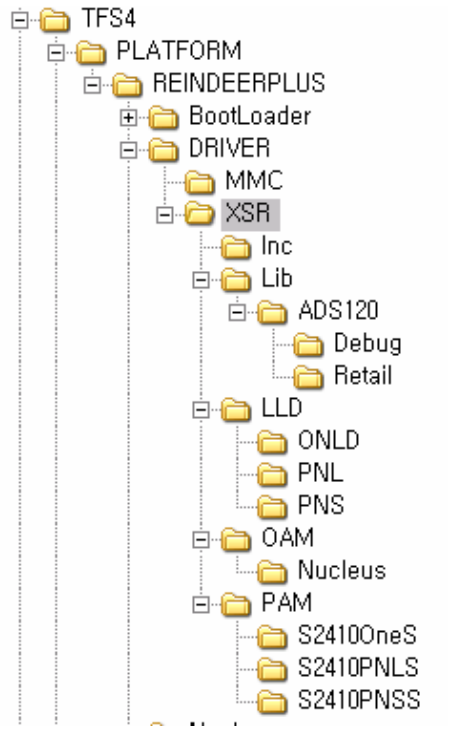


Figure 3-4. XSR Directory Structure

Currently, the above indicates the directory structure of the XSR version 1.5.0. The following table describes the XSR directories shown in the above picture.

Table 5. Description of XSR Directories

Directory	Contains
XSR	XSR Library and API headers
XSR \LLD\ONLD XSR \LLD\PNL XSR\LLD\PNS	Low Level Device Driver
XSR \OAM\Nucleus	Adaptation code for Nucleus
XSR \PAM\S2410OneS XSR \PAM\S2410PNLS XSR \PAM\S2410PNSS	Platform Adaptation Module
XSR \lib\ADS120\Retail, Debug	Retail, Debug version XSR Library(ARM Compiler, ADS v1.2)

XSR \Inc	Header files for XSR

3.3. Directory Structure and Descriptions of MMC (or HSMMC) Host Device Driver

If you want to support a removable external device such as MMC, you must write a device driver for that device. At the "TFS4\PLATFORM\REINDEERPLUS\DRIVER\MMC," you can find an example MMC device driver for REINDEERPLUS.
For more details about writing a device driver for the removable external device, refer "II. MMC (or HSMMC) Host Device Driver APIs"

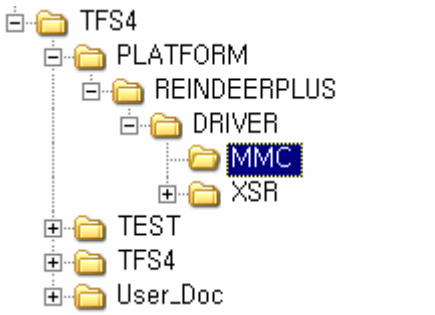


Figure 3-5 MMC Device Driver Source Directory

4. TFS4 Porting Procedure

This chapter describes TFS4 porting procedure in detail. The procedure is divided into five steps as the following picture.

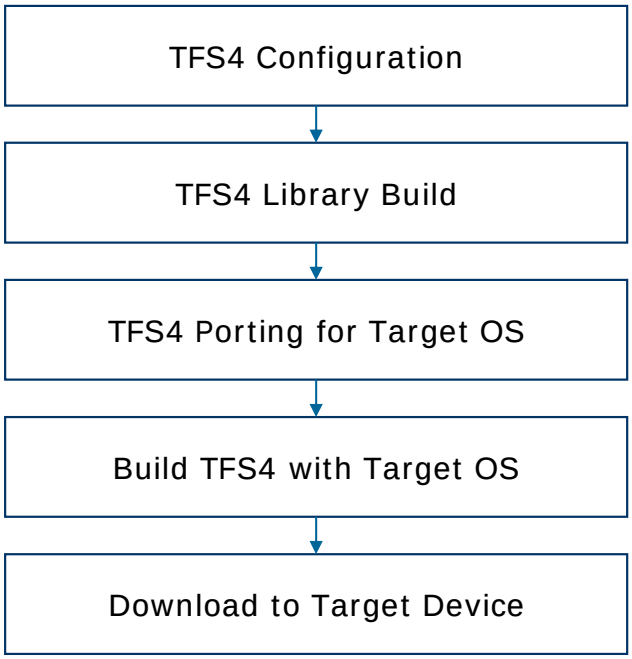


Figure 4-1. TFS4 Porting Procedure

You don't have to follow the above steps exactly. The build process is up to you. If you do not want to build TFS4 library first and just want to build all TFS4 components together, right before TFS4 build with OS, you can do in that way.

The above sequence is a kind of guideline for porting TFS4.

4.1. TFS4 Library Build

This section describes TFS4 library build procedure. You can first configure TFS4 configuration files and build the sources.

4.1.1. TFS4 Configuration

There are three files for TFS4 configurations:

- tfs4_config_base.h
- tfs4_config_const.h

If you need to modify the TFS4 configurations, refer to the description of each file, and configure them adequate to your target environment.

4.1.1.1. tfs4_config_base.h

tfs4_config_base.h should be modified according to your target environment. If you configure the tfs4_config_base.h file, you should re-build the TFS4 libraries.
tfs4_config_base.h is in the "C:\TFS4\TFS4\BASE\INC" directory.
tfs4_config_base.h has configuration entries for OS and base library such as UNICODE, time and random functions.

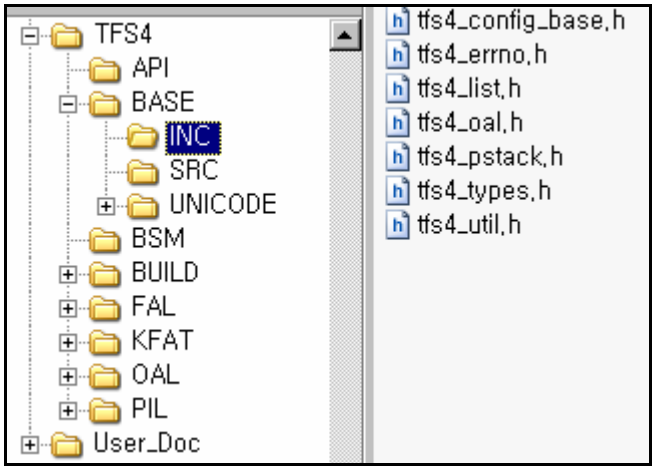


Figure 4-2. tfs4_config_base.h

- TFS4_OS

It specifies target OS. There are `TFS4_NUCLEUS`, `TFS4_PSOS`, and `TFS4_UNKNOWN` in the TFS4_config_const.h file. The currently available OSs are `TFS4_NUCLEUS` and `TFS4_PSOS`. `TFS4_WIN32` and `TFS4_LINUX` are used for development and test. Specially, Nucleus was used for test on the REINDEERPLUS.

The following shows the configuration of Nucleus OS in the tfs4_config_base.h file as a sample.

```
//// Configuration section #1
/*
    1. Define Operating System.
        TFS4_WIN32, TFS4_NUCLEUS, TFS4_PSOS, TFS4_RTKE,
        TFS4_LINUX, TFS4_REX, TFS4_VXWORKS
    are supported now
*/
#define TFS4_YOUR_OS
```

Figure 4-3. tfs4_config_base.h

If your target OS is not listed here, then define a new OS and modify TFS4 source files which depend on the target OS. For defining a new target OS, refer the other example.

- TFS4_UNICODE

It specifies TFS4 works with UNICODE. Normally TFS4 receive multi-byte string as input. But if TFS4_UNICODE is defined, TFS4 uses UNICODE (UTF-16) string for path name and other string.

- TFS4_CODEPAGE

It specifies the default language code page for TFS4. The supported code pages are listed at 'Appendix. VIII. Code Pages'. It affects the creation of the short filename from the given long filename. As TFS4 supports the FAT filesystem, and the FAT filesystem needs to use the codepage, you must specify which codepage you would use.

- TFS4_BYTE_ORDER

It specifies a byte ordering of your target. If your target uses little endian, you can set `TFS4_LITTLE_ENDIAN` as follows. You can define it as follows.

```
#define TFS4_BYTE_ORDER  TFS4_LITTLE_ENDIAN
```

If your target uses big endian, you should set TFS4_BYTE_ORDER to TFS4_BIG_ENDIAN.

- TFS4_HAS_TM

It specifies whether your target system compiler supports '`struct tm`' data type (C standard). If your target system compiler does not support the standard '`struct tm`', you can comment it out or set to '0'.

You need to include header files for your target OS and define the value of true and false.

- Headers and base definitions for the target OS and the target compiler

If your compiler does not support an inline function nor has a different format of inline function, you have to change the inline setting in the tfs4_config_base.h file as follows.

```
#if (TFS4_OS == TFS4_WIN32)
#include <fcntl.h>
#include <time.h>
#include <windows.h>
#define true   1
#define false  0
#define inline  __inline

#elif (TFS4_OS == TFS4_LINUX)
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <stdlib.h>
      #ifdef TFS4_WATCOM
            #define inline
      #endif

#elif (TFS4_OS == TFS4_NUCLEUS)
#include <time.h>
#include <stdlib.h>
#define inline
```

```
#if   (__CC_ARM == 1)               // Check Use ADS1.x
      #define TFS4_HAS_STDARG_H        1
#endif

#elif (TFS4_OS == TFS4_PSOS)
#include <types.h>
#include <fcntl.h>
#include <time.h>
#define true   1
#define false  0
#define inline

#elif(TFS4_OS == TFS4_RTKE)
#include <time.h>
   #include "din4rtkg.hec"
#define true      1
#define false     0
#define inline

#define TFS4_FILE_OPEN_FLAG
   #ifdef TFS4_FILE_OPEN_FLAG

/*********************************************************/
/* file access-mode, creation and status flags used with
open()/fcntl() */
/*********************************************************/
      #define O_RDONLY   0x0000  /* Open for reading only */
      #define O_WRONLY   0x0001  /* Open for writing only */
      #define O_RDWR     0x0002    /* Open for reading and
      writing */
      #define O_NONBLOCK 0x0004  /* No delay */
      #define O_RAWMEM   0x0008  /* ISI: reserved for use by
      pSE+ */

      #define O_EXCL     0x0100  /* Exclusive use flag */
      #define O_CREAT    0x0200  /* Create file if it does not
      exist  */
      #define O_NOCTTY   0x0400  /* Do not assign a controlling
      term.*/
      #define O_TRUNC    0x0800  /* Truncate flag */

      #define O_APPEND   0x0010  /* Set append mode */
      #define O_SYNC     0x0020  /* Write accd. to SIO file
      integrity */
      #define O_DSYNC    0x0040  /* Write accd. to SIO data
      integrity */
      #define O_RSYNC    0x0080    /* Synchronized read I/O
      operation */

      #define O_ACCMODE  0x0003  /* Mask for file access mode
      */
   #endif

#endif
```

4.1.1.2. tfs4_config_const.h

tfs4_config_const.h should be modified according to your target environment. If you configure the tfs4_config_const.h file, you should re-build the TFS4 libraries. tfs4_config_const.h is in the "C:\TFS4\TFS4\FAL\INC" directory on your host as follows.

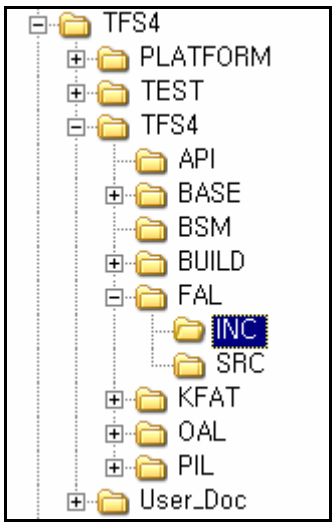


Figure 4-4 tfs4_config_const.h

- TFS4_WRITE_ACCELERATE

It is a flag to write only the first log when some data is appended within the same cluster. It improves performance for which append operation of small data occurs frequently. You can set the value as 1 to enable the feature. Or, set the value as 0 to disable the feature.

- TFS4_INIT_CLUSTER

It specifies whether a newly assigned cluster from FAT is initialized as 0 or not. If the value is "1," the cluster is initialized as 0. Initializing as 0 is POSIX standard. But, it decreases the write performance. Disable it on your own risk.

- TFS4_USE_EXT_INTERFACE

It is needed when you want perfect compatibility with MS FAT filesystem.
TFS4 stores some data at file and directory metadata area to improve the performance of lookup operation. But it does not guarantee full compatibility with MS FAT filesystem.

Note. TFS4 does not store any additional meta data on MMC (or HSMMC) media even if this definition does not exist.

- TFS4_FAST_SEEK

It specifies whether the file system uses the fast seek functionality that improves file pointer movement performance. If you want to support the fast seek, TFS4_FAST_SEEK has to be defined as '1'. You have to specify this definition to 1 to use tfs4_fast_seek() function. For using fast seek, refer to TFS4 Programmer's Guide.

- TFS4_FILE_LEVEL_FLUSH

It specifies whether the file system uses the file level flush that improves tfs4_fsync() performance. If you want to support the file level flush, TFS4_FILE_LEVEL_FLUSH has to be defined as '1'.

- TFS4_HIDDEN_DIR

It specifies whether the file system uses hidden area or not. If you want to support the hidden area, TFS4_HIDDEN_DIR has to be defined as '1'.
The hidden area is a TFS4 specific directory that is not shown on other FAT compatible filesystem. It is a special directory that can be listed by filesystem. But user can access the directory with pre-defined path like a normal directory.
User must use tfs4_ioctl() to control access to the hidden area. Refer to the TFS4 Programmer's Guide. There is another configuration for "TFS4_HIDDEN_DIR" at the next chapter 4.1.1.3 tfs4_config.h

- TFS4_RESCUE

This constant enables the APIs, tfs4_restore() and tfs4_backup(), that are used for saving file system metadata and restoring a file system from it. It is originally intended for magnetic disks that are error-prone.

If you want to use the APIs, set '1'. If unsure, set '0'.

- TFS4_FILE_LOCKING

File locking provides the applications a method to get exclusive access rights to an open file. If enabled, you can manages the access permission on a specific file through tfs4_fcntl(). For more detailed information, please refer to the TFS4 Programmer's Guide.

If you want to use it, set '1'. Otherwise, set '0'.

- TFS4_BAD_SECTOR_MANAGER

You can enable or disable the bad sector manager by adjusting this constant. TFS4 provides a filter driver layer, called BSM that wraps low-level physical device driver and handles bad sectors on the device. It reserves storage spaces for reallocating bad sectors and redirects I/O requests to this area. So this may be dangerous when used without a full understanding.

To add this feature, set '1'. If you don't need to use the BSM or don't know exactly what it is, it is strongly recommended to set '0'.

- TFS4_FAST_UNLINK

This option allows you to delete a large file extremely fast by using the Fast Unlink technology introduced from TFS4 v1.5. It attaches the deleted cluster chain to a pool file named 'delete.me' instead of clearing the entire FAT links. But, the system file, 'delete.me' unwanted will be created. The other thing you should aware is that the REAL free space of the volume would not be recovered immediately after unlinking a file, but it does not matter on creating a new file as TFS4 recycles the space occupied by 'delete.me'. Unlinking the pool file, 'delete.me' is not allowed using tfs4_unlink() and it is an intended operation.

To add this feature, set '1'. If your files are usually small, it is recommended to set '0'.

NOTICE!!

Fast Unlink may decrease append write performance of TFS4. It is caused by some additional operation for updating the pool file (‘delete.me’).
The following is a method to remove this performance down. We recommend this work while in an idle state. It makes the pool file to a small size.

1. Create and open a temporary file with flag
   “O_RDWR | O_TFS4_NORMAL_UNLINK | O_TFS4_NO_INIT_CLUSTER“.
2. Truncate the file to a big size over 16KB.
3. Truncate the file to size zero.
4. Re-do operations 2 and 3 until the pool file size is 0.
5. Close and unlink the temporary file.

- TFS4_LOG_FILE_NAME

It specifies a log file name for TFS4. TFS4 writes log for recovery from abnormal operation. TFS4 creates a log file in the root directory on each volume. You can specify the log file name on your purpose. If you use UNICODE, you also have to specify the log file name as a UNICODE string according to endian.

- TFS4_ATTR_NEW

It is needed when you want to use user attribute support. The two user attributes are TFS4_ATTR_USR1 and TFS4_ATTR_USR2. You can use these attributes on your purpose. These attributes can be used in tfs4_stat(), tfs4_stat_set(), tfs4_fstat(), tfs4_fstat_set(). But these attributes can not be used on external device such as MMC.

NOTICE !!

This definition may make some compatibility problem with another FAT compatible file system. These two attributes use an area which is not used in FAT specification, but another File System can use the area too.

This definition does not be used with TFS4_USE_EXT_INTERFACE definition.

- TFS4_DIR_NAME_MAX_LENGTH

It specifies the maximum length of the directory name. For Windows, it is 243 at maximum.

- TFS4_FILE_NAME_MAX_LENGTH

It specifies the maximum length of the file name. For Windows, it is 255 at maximum including an extension.

- TFS4_PATH_NAME_MAX_LENGTH

It specifies the maximum path length. For example, the path length of /a/dir1/test.txt is 16.

- TFS4_PATH_COMPONENT_MAX

It specifies the maximum number name of directory and file on path.

- TFS4_MAX_DEVICE_NAME_LENGTH

It specifies the maximum length of device name. Do not change this value.

- TFS4_MAX_VOLUME_NAME_LENGTH

It specifies the maximum length of volume name. Do not change this value.

- TFS4_MAX_FILE_SYSTEM_NAME_LENGTH

It specifies the maximum length of filesystem name.

- TFS4_MAX_FILE_LENGTH

It specifies the maximum size of a file. The default value is 4 byte of signed integer.

- TFS4_SSIZE_MAX_LENGTH

It specifies the maximum data size for a read or write operation.

- TFS4_MAX_PAGE_SIZE

It specifies the maximum page size of physical device. You don't have to change it. It is normally 4.

- TFS4_DEBUG

It is needed when you build TFS4 with debug mode.

- TFS4_FILESYSTEM_MAX

It specifies the maximum number of file systems that your target supports.

- TFS4_NAME_MAX_LENGTH

It specifies the maximum number of bytes for file and directory name. Considering Unicode, 510 bytes is the maximum.

- TFS4_VOLUME_NAME_LENGTH

It specifies the volume name length. It is 3, because volume name is shown as /a/, /b/, etc.

- TFS4_DEVICE_NAME_LENGTH

It specifies the device name length. It can be up to 8, because device name is shown as /dev/nf0, /dev/nf1, /dev/mmc0, etc.

- TFS4_SECTOR_SIZE

It specifies the sector size in byte.

- TFS4_SECTOR_SIZE_BITS

It specifies the number of bits for indicating `TFS4_SECTOR_SIZE`.

- TFS4_SECTOR_SIZE_MASK

It is used for fast operation. You specify it as ((1 << `TFS4_SECTOR_SIZE_BITS`) - 1).

4.1.1.3. tfs4_config.h

tfs4_config.h is the configuration file of TFS4. It can be automatically applied to TFS4 if you modify the tfs4_config.h file. tfs4_config.h does not affect the TFS4 library and you don't have to re-build the TFS4 library, even though tfs4_config.h is modified.

tfs4_config.h is on "C:\TFS4\TFS4\FAL\INC" directory.

The following shows the configurations in tfs4_config.h.

```
/****************************************************************************
#ifndef __TFS4_CONFIG_H__
#define __TFS4_CONFIG_H__

#include <string.h>
#include <stdio.h>

#include "tfs4_config_const.h"

#define TFS4_PDEV_COUNT     (3)     //// physical device count max
#define TFS4_VOLUME_COUNT   (8)     //// volume count max

//// File System Configuration
#define TFS4_FILE_MAX         32  //// maximum number of file entry
#define TFS4_FILE_OPEN_MAX    48  //// maximum number of concurrently open fi
#define TFS4_MAX_DIR_OPEN     16  //// maximum number of concurrently open di

#define TFS4_FILE_HASH        9   //// FILE list hash base

// buffer cache
#define TFS4_CACHE_COUNT                128         //// buffer count, shoul
#define TFS4_BCACHE_HASH_VALUE          TFS4_CACHE_COUNT
#define TFS4_BCACHE_HASH_MASK           (TFS4_CACHE_COUNT - 1)

#define TFS4_BCACHE_READ_AHEAD          //// enable CACHE-READ-AHEAD
#define TFS4_BCACHE_READ_AHEAD_COUNT    8
#define TFS4_BCACHE_READ_AHEAD_MASK     (TFS4_BCACHE_READ_AHEAD_COUNT - 1)

#define TFS4_BCACHE_WRITE_BACK          //// enable WRITE-BACK policy
#define TFS4_BCACHE_WRITE_BACK_COUNT    8

//// For direct I/O
#define TFS4_DIRECT_IO_SECTOR           ((TFS4_CACHE_COUNT >> 2) + TFS4_CACH

//// FAT cache
#define TFS4_FS_FAT_CACHE_SIZE          32                                  ////
#define TFS4_FCACHE_HASH_VALUE          TFS4_FS_FAT_CACHE_SIZE              ////
#define TFS4_FCACHE_HASH_MASK           (TFS4_FCACHE_HASH_VALUE - 1)    /* d

//// PATH (directory) cache
#define TFS4_PATH_CACHE                 //// enabel path cache
#define TFS4_PATH_CACHE_COUNT           16
#define TFS4_PATH_CACHE_HASH_VALUE      16
#define TFS4_PATH_CACHE_HASH_MASK       (TFS4_PATH_CACHE_HASH_VALUE - 1)
#define TFS4_PATH_CACHE_COUNT_TO_FREE   (TFS4_PATH_CACHE_COUNT >> 2)
```

Figure 4-5. tfs4_config.h

- TFS4_PDEV_COUNT

It specifies the number of physical devices that your target supports. This is the count of physical devices that can be registered with tfs4_pdev_reg()

- TFS4_VOLUME_COUNT_MAX

It specifies the maximum number of logical devices that your target supports. It is same as the value of `TFS4_LDEV_COUNT`. The maximum value is 26.

- TFS4_VOLUME_COUNT

It specifies the maximum number of volumes that can be mounted concurrently. The maximum value is 26.

- TFS4_LDEV_COUNT

It specifies the maximum number of partitions that can be mounted concurrently. This value is same as the value of TFS4_VOLUME_COUNT.

- TFS4_FILE_MAX

It specifies the maximum number of different files that you can open. Whenever each volume is mounted, the number of files that you can open is decreased by 1, because the root directory is opened for each mounted volume. TFS4_FILE_MAX determines the total number of different files and directories that can be opened.

- TFS4_FILE_OPEN_MAX

It specifies the maximum number of files that you can open. But it includes the number of files re-opened. Therefore, it should be larger than TFS4_FILE_MAX.
TFS4_FILE_OPEN_MAX determines the total number of files and directories that can be opened at once.

- TFS4_MAX_DIR_OPEN

It specifies the maximum number of directories that you can open simultaneously.

The above TFS4_FILE_MAX, TFS4_FILE_OPEN_MAX, and TFS4_MAX_DIR_OPEN settings are needed, because some same files can be opened more than 2 times.

For example, if A opens "/a/readme.txt", it uses each one resource from a TFS4_FILE_MAX and TFS4_FILE_OPEN_MAX. If B opens "/a/readme.txt", it uses one resource from a TFS4_FILE_OPEN_MAX. If C opens "/a/mydir/" by a tfs4_opendir(), it uses one resource from each TFS4_FILE_MAX, TFS4_FILE_OPEN_MAX, and TFS4_MAX_DIR_OPEN.

- TFS4_FILE_HASH

It specifies the hash length of file table. You can set it as the odd number or decimal number near TFS4_FILE_MAX / 3.

- TFS4_CACHE_COUNT

It specifies the number of cache blocks (in sector unit), which is used by Buffer cache manager. Each cache has the size of TFS4_SECTOR_SIZE. It should be set to memory usage and more than 8 at minimum.

- TFS4_BCACHE_HASH_VALUE

It specifies the HASH length. You can set this as TFS4_CACHE_COUNT.

- TFS4_BCACHE_HASH_MASK

It is used for fast operation. You specify it as (TFS4_CACHE_COUNT - 1).

- TFS4_BCACHE_READ_AHEAD

If you define TFS4_BCACHE_READ_AHEAD, read ahead operation is performed on buffer cache.

- TFS4_BCACHE_READ_AHEAD_COUNT

It specifies the number of sectors for the read ahead operation on buffer cache.

- TFS4_BCACHE_READ_AHEAD_MASK

It is used for fast operation. You specify it as (TFS4_BCACHE_READ_AHEAD_COUNT - 1).

- TFS4_BCACHE_WRITE_BACK

If you define TFS4_BCACHE_WRITE_BACK, write operation stores data at the buffer cache and the write operation is delayed. The data may be lost due to the sudden power off.

- TFS4_BCACHE_WRITE_BACK_COUNT

It specifies the number of buffers for the write back operation.

- TFS4_DIRECT_IO_SECTOR

It specifies the minimum number of sectors for write operation with no cache.

- TFS4_FS_FAT_CACHE_SIZE

It specifies the number of sectors for caching.

- TFS4_FCACHE_HASH_VALUE

It specifies the HASH length.

- TFS4_FCACHE_HASH_MASK

It is used for fast operation. You specify it as (TFS4_FCACHE_HASH_VALUE - 1).

- TFS4_PATH_CACHE

It enables the path cache function.

- TFS4_PATH_CACHE_HASH_VALUE

It specifies the HASH value of the path cache.

- TFS4_PATH_CACHE_HASH_MASK

It is used for fast operation. You specify it as (TFS4_PATH_CACHE_HASH_VALUE - 1).

- TFS4_PATH_CACHE_COUNT_TO_FREE

It specifies the number of path cache entries to free for adding a new entry, when the path cache is full. For example, if the total number of path cache entries is 10, all they are used, TFS4_PATH_CACHE_COUNT_TO_FREE is set as 3, and a new entry has to be added, 3 entries would be freed from the total 10 entries and new entry would be added.

- TFS4_HIDDEN_DIR_NAME

It specifies the name of hidden directory. They are not be shown on other FAT compatible file system that all of the files and directory and hidden directory itself below this directory. But TFS4 can access these entries after the volume is mounted with TFS4_MOUNT_HDIR(refer to the TFS4 Programmers Guide, tfs4_mount()). The maximum length of the name is 26 byte.

- TFS4_HDIR_MAX_VOLUME_SIZE

It specifies the maximum volume size can be used for hidden area. It is specified with Mega-Byte.

- TFS4_HDIR_VOLUME_COUNT

It specifies the maximum volume count can be mounted for hidden area concurrently.

4.1.2. TFS4 Library Build

This section describes how to build TFS4 source files of which a filesystem component is composed. You can select a build tool considering your target environment, such as makefile, ADS, or Code Composer.

TFS4 sources can be divided into two parts;
- Sources independent of the target & tfs4_config.h

- Sources dependent on the target & tfs4_config.h

You can first build the independent sources to your target and tfs4_config.h file and make into a library file, because they are not related to target and OS configuration. The following figure shows the TFS4 source files to make a library.

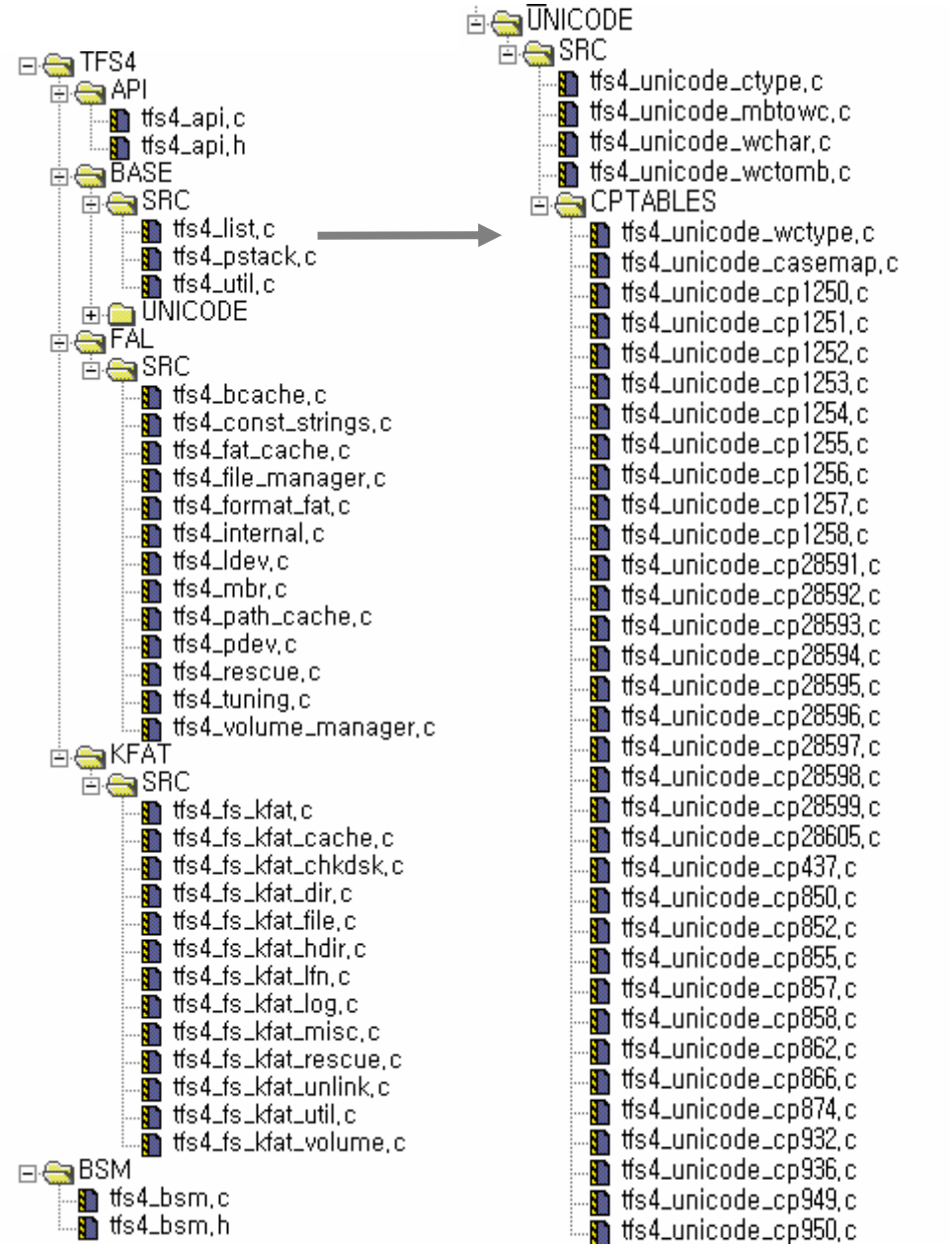


Figure 4-6. Source files for TFS4 Library

The above source files are all independent source files of your target environment. You don't need to re-build them, even if tfs4_config.h is modified. But, they should be built again if tfs4_config_const.h, tfs4_config_base.h or the compile environment is changed.
For example, if you modify MMC(or HSMMC) device setting, byte order, maximum

length of directory and file in the tfs4_config_base.h and the tfs4_config_const.h, it affects all the components of TFS4, so you must re-build the TFS4 sources.

When the number of cache is modified for TFS4 tuning, you don't have to build the TFS4 library again and consequently it reduces a build time.

< Building TFS4 on ADS v1.2 >
This is the build steps of TFS4 sources.

1. Execute your build tool. ADS v1.2 (Metrowerks CodeWarrior for ARM Developer Suits v 1.2) is used in this document.

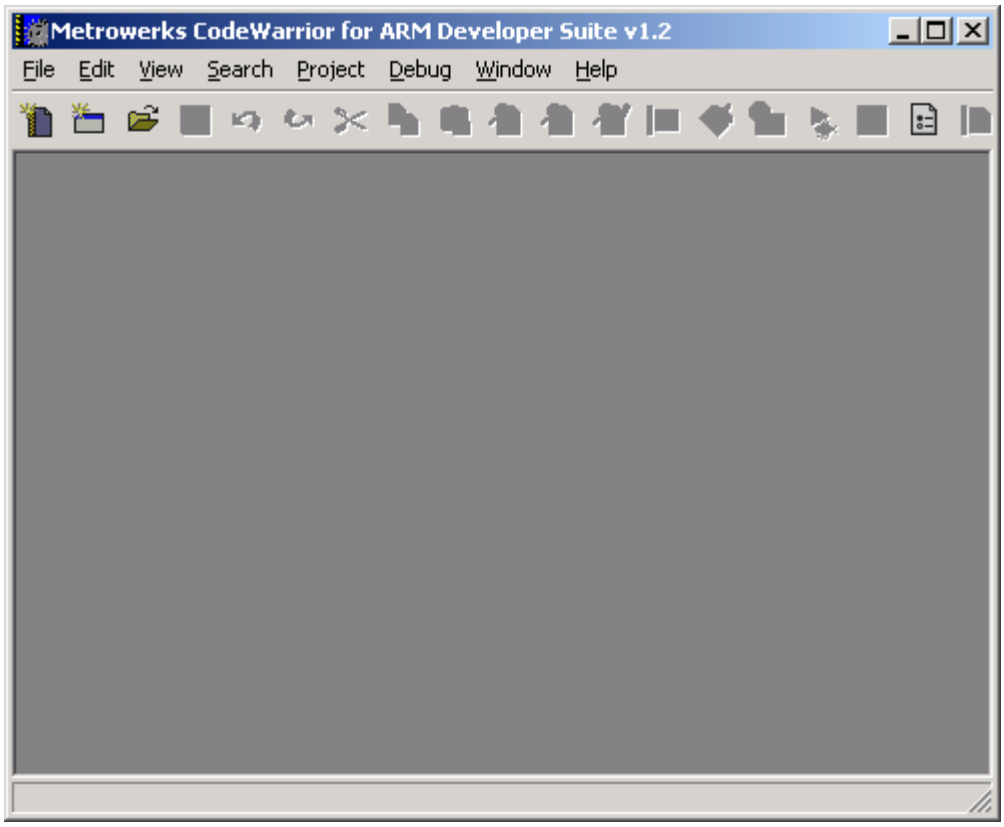
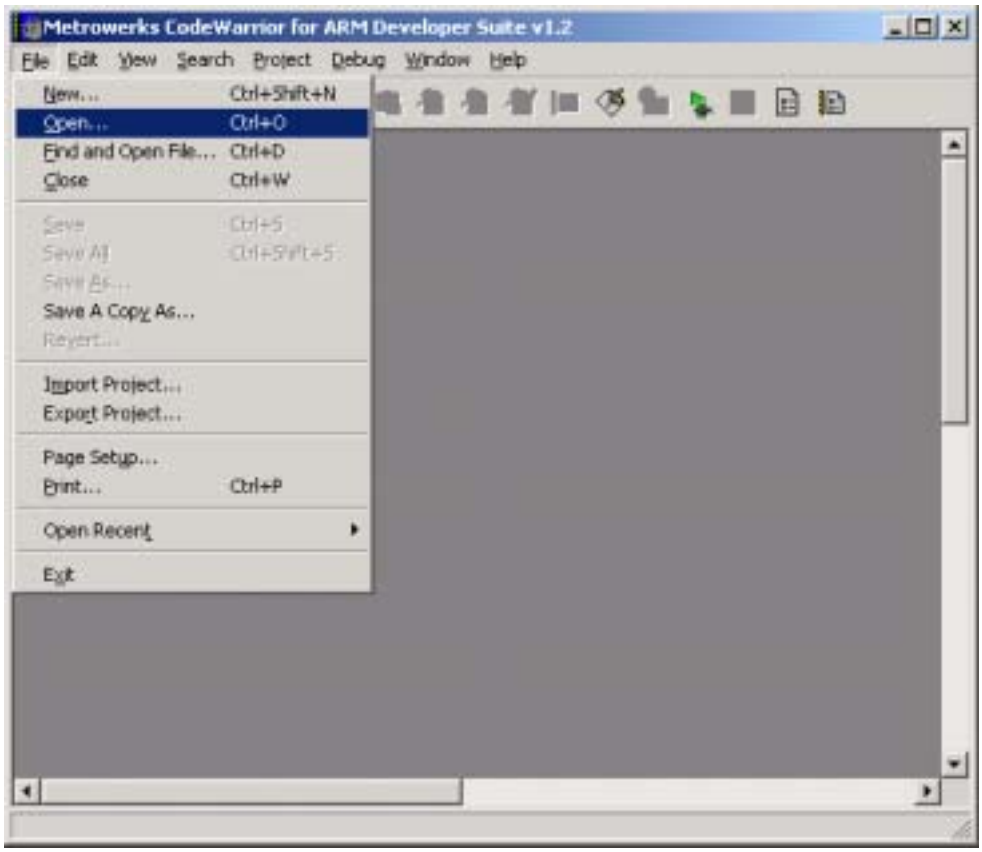


Figure 4-7. ADS v1.2 Initial Screen

2. Open the project file for making a TFS4 library. You can click "File" → "Open" on the menu bar of the screen as follows.

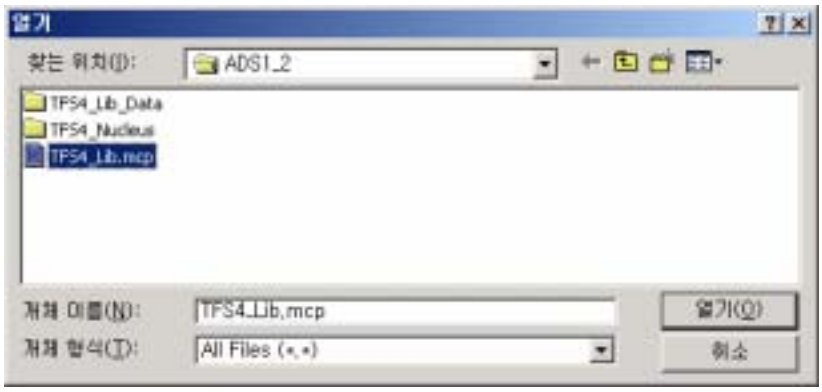


Or,  button on icon bar.

Note

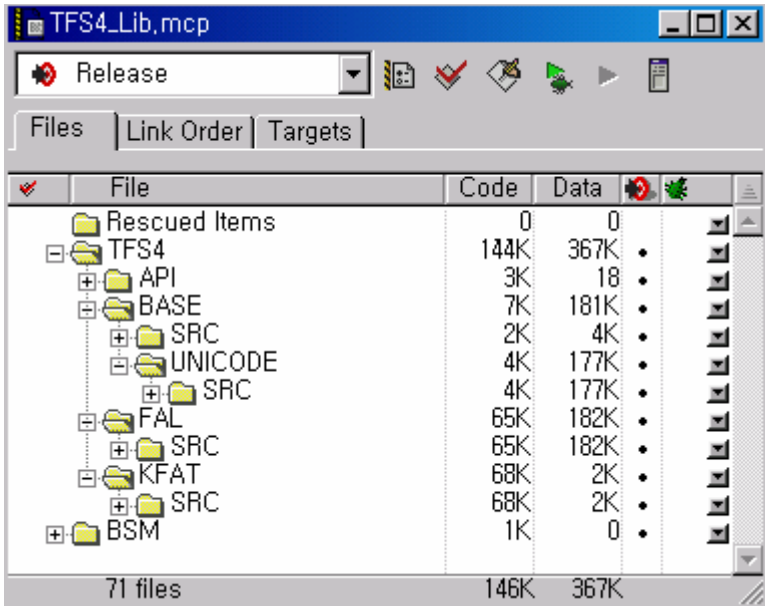
We provide a sample project file for building TFS4, a TFS4_Lib.mcp; the extension “mcp” is the project file extension of ADS build tool. If you don’t use the ADS v1.2, you need to create a project file on your build tool and add the TFS4 source files to the project file.

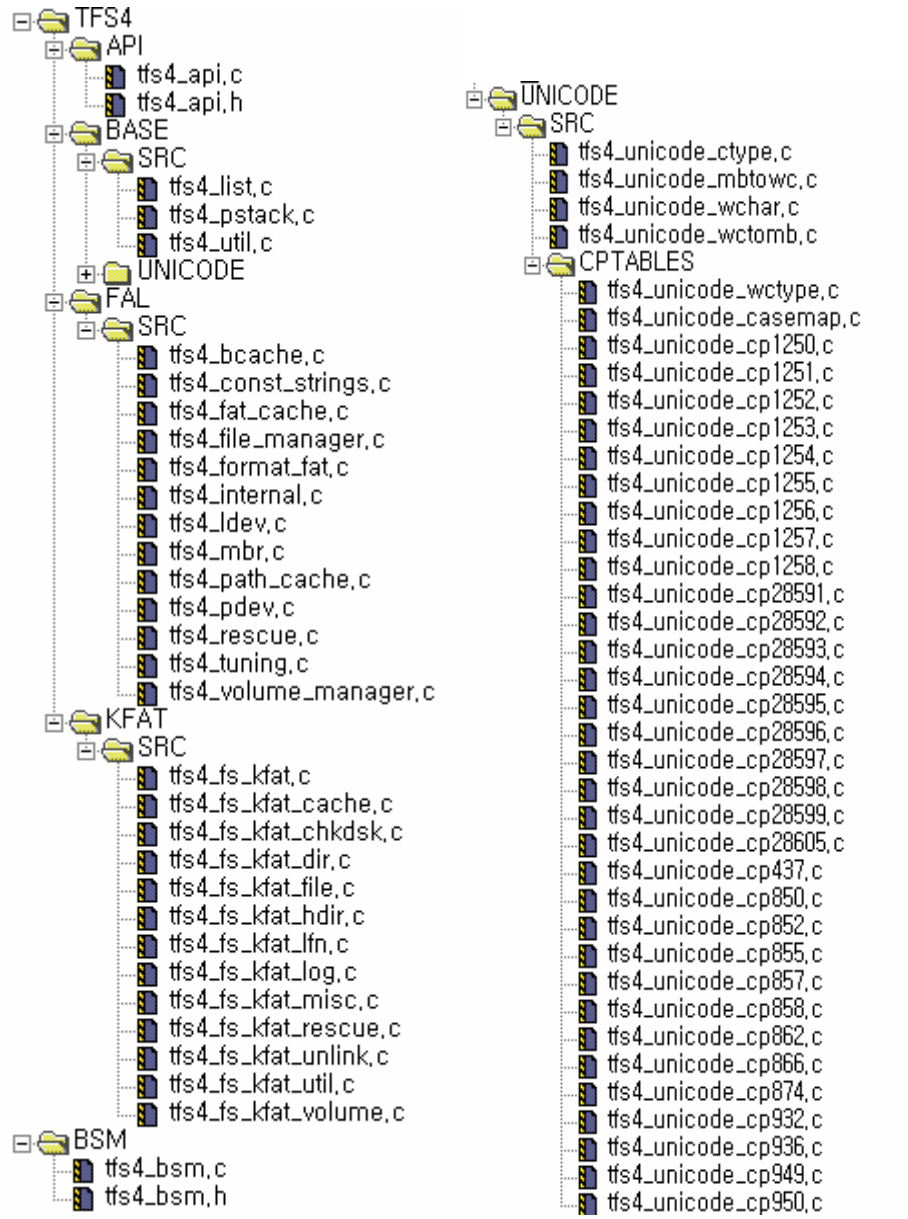
3. The screen to open a file appears.



Find the TFS4_Lib.mcp file and press “Open” button on the screen.

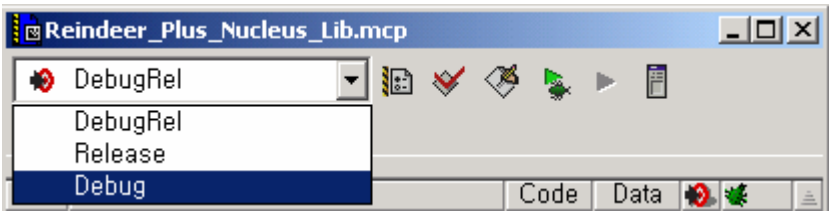
4. The TFS4_Lib.mcp file is opened as below.





See the TFS4 source and header files to build in the project file.
There are many tfs4_unicode_cpxxx.c on the right figure. But you do not need to add all of them to the project. You add just only one file on your TFS4_CODEPAGE configuration at tfs4_config_base.h.

5. Select a type of build target..



There are three types of build targets. The following describes the meaning of each build target.

- Debug: The output binary is compiled with debugging symbols and information of line numbers.
- Release: In this configuration, the output binary will be fully optimized and contains no debugging symbols.
- DebugRel: Adequate optimization level and including minimal debugging information.

6. Press  the build setting button.

7. The build settings screen shows up.

The build setting screen can be a little different according to the build mode setting you select. You have to consider the language setting your compiler supports on the build setting screen. For ADS v1.2, you need to set the build options each language; you may not need to do that for other build tool.

The following screens show the build options needed for each build mode and language; the sample build options are based on ReindeerPlus, ARM CPU, and ADS v1.2. You can set the build options suitable to your target and compile environment by referring to the below sample options.

Note

The build options depend on the compiler. This TFS4 build section only explains the options related to TFS4: TFS4_NUCLEUS, TFS4_KFAT, and TFS4_DEBUG (it is optional for debug mode).

But if they are already defined in the tfs4_config_const.h file, you don't have to enter the options for compiling here.

<Debug settings for ARM Assembler>

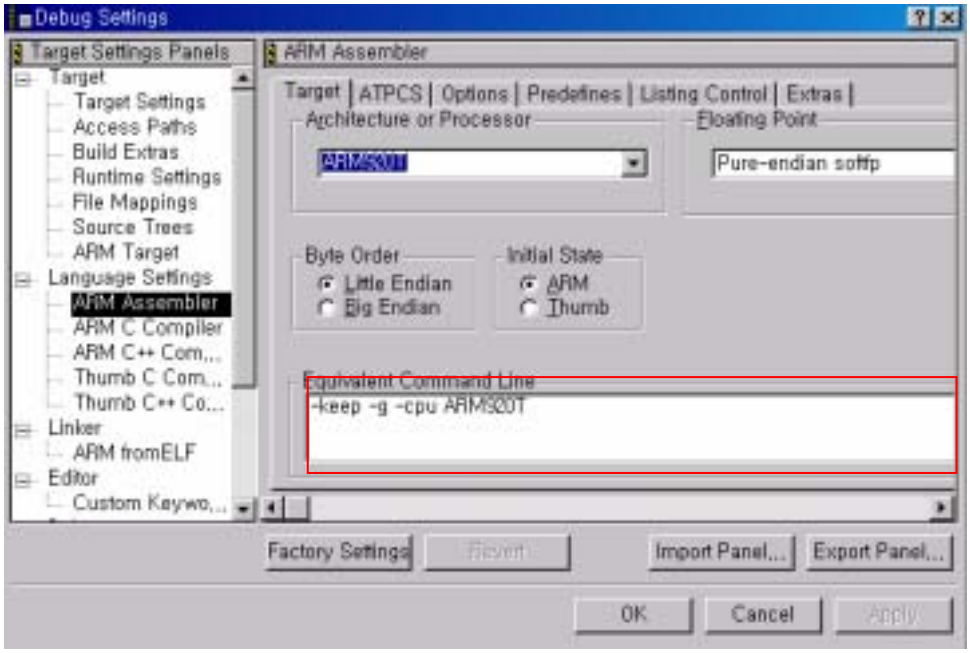


Figure 4-8. Debug Settings for ARM Assembler on ADS v1.2

<Debug settings for ARM C Compiler>

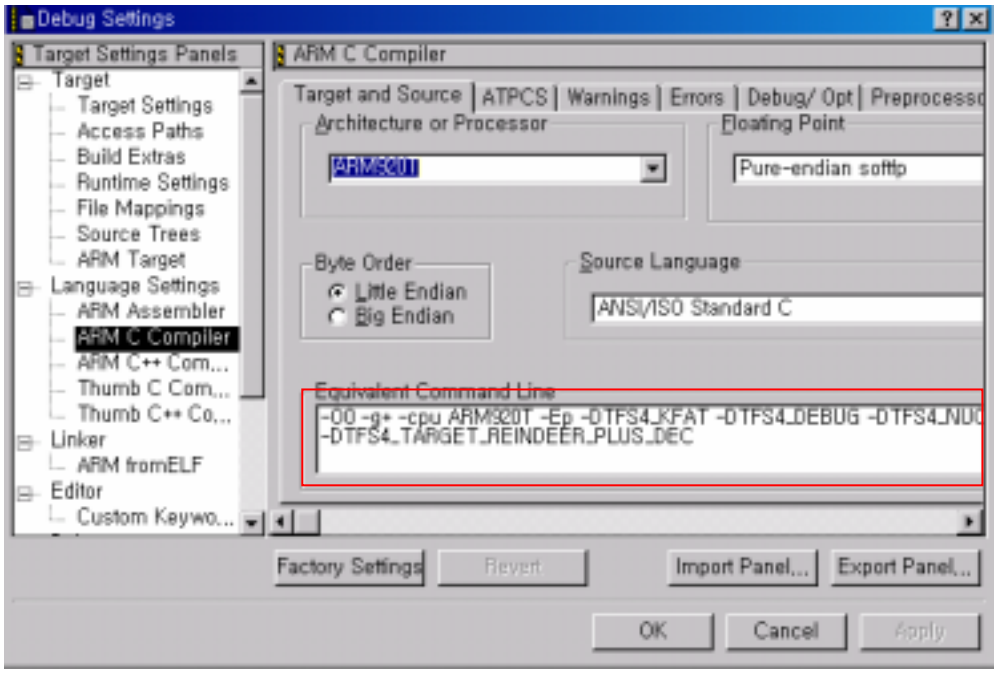


Figure 4-9. Debug Settings for ARM C Compiler on ADS v1.2

<Debug settings for ARM C++ Compiler>

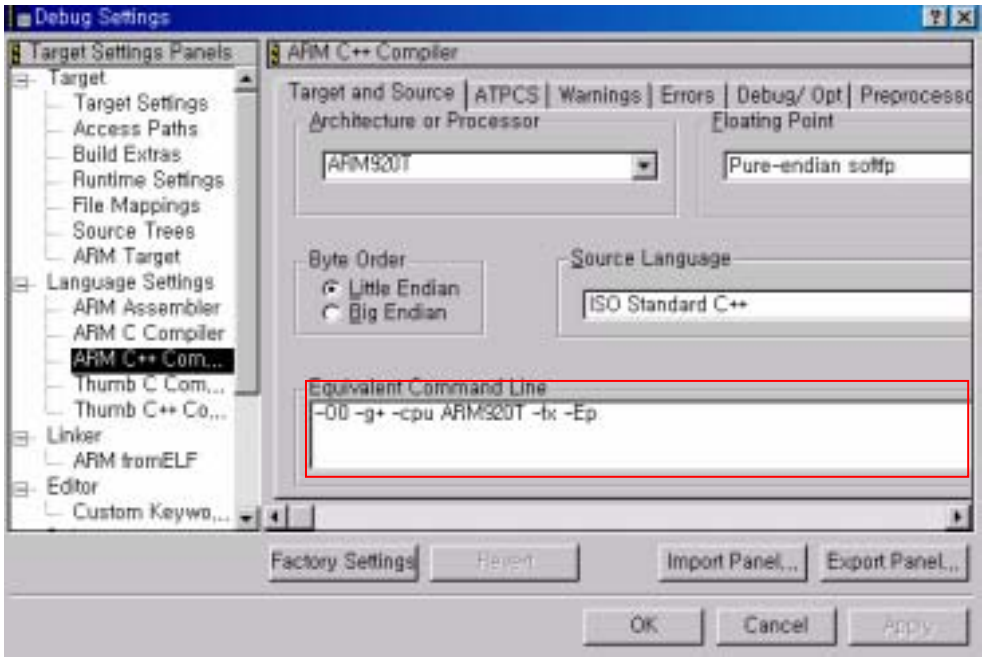


Figure 4-10. Debug Settings for ARM C++ Compiler on ADS v1.2

< Release settings for ARM Assembler>

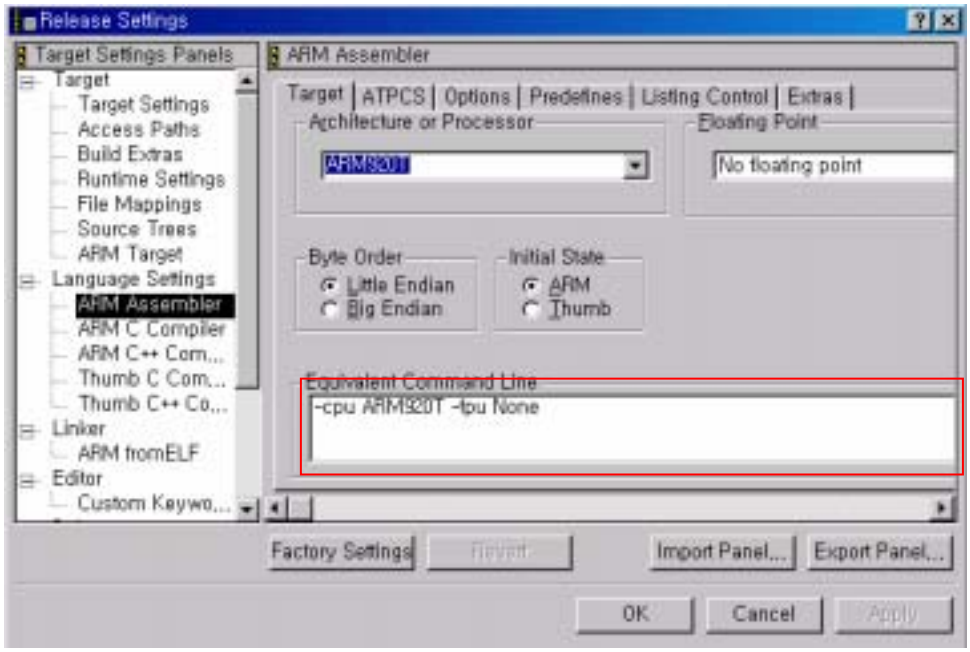


Figure 4-11. Release Settings for ARM Assembler on ADS v1.2

< Release settings for C Compiler>

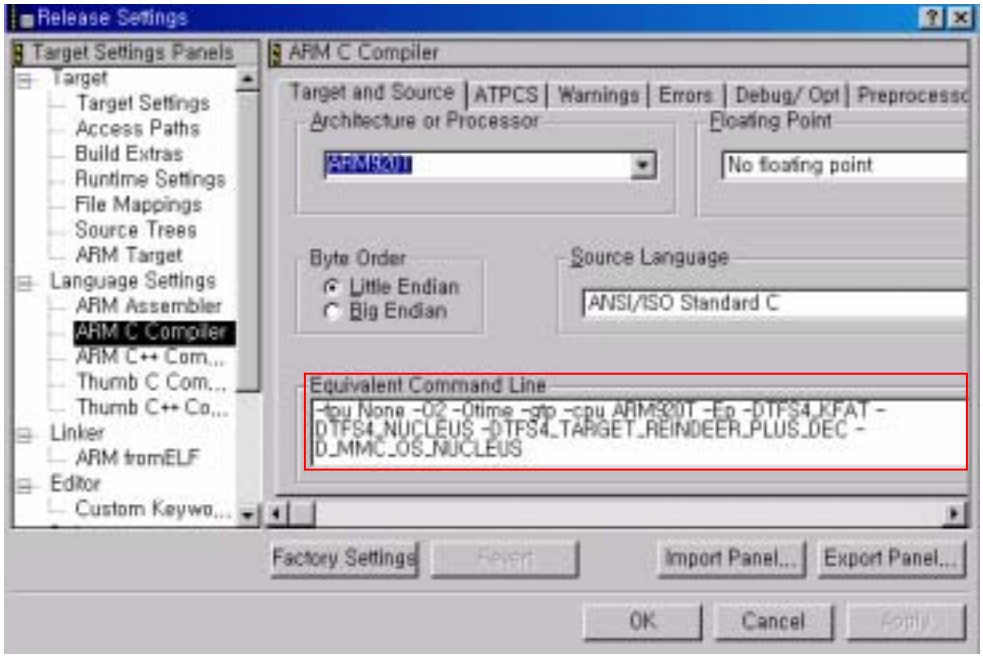


Figure 4-12. Release Settings for ARM C Compiler on ADS v1.2

< Release settings for C++ Compiler>

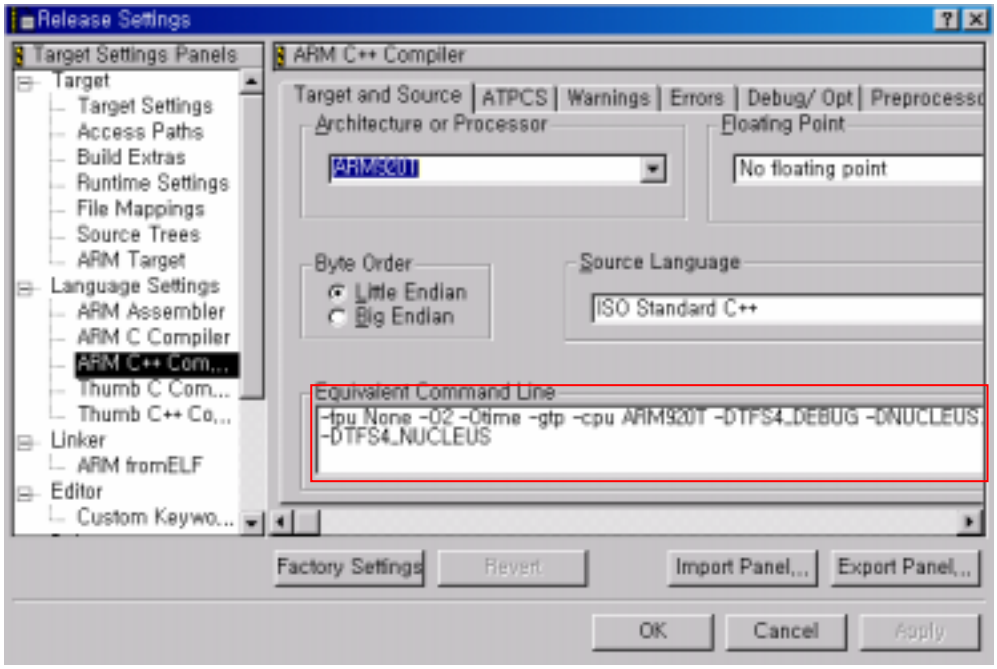


Figure 4-13. Release Settings for ARM C++ Compiler on ADS v1.2

You have to add the build options for your target and compile environment.

8. Additionally, you need to add the access paths on the build settings to include the TFS4-related header files (TFS4\API, TFS4\BASE\INC, TFS4\BASE\UNICODE\INC, TFS4\FAL\INC, TFS4\KFAT\INC, TFS4\OAL\NUCLEUS\INC, TFS4\PIL\INC, XSR header file path, and MMC(or HSMMC) host device driver header file path, etc) while TFS4 is compiled.

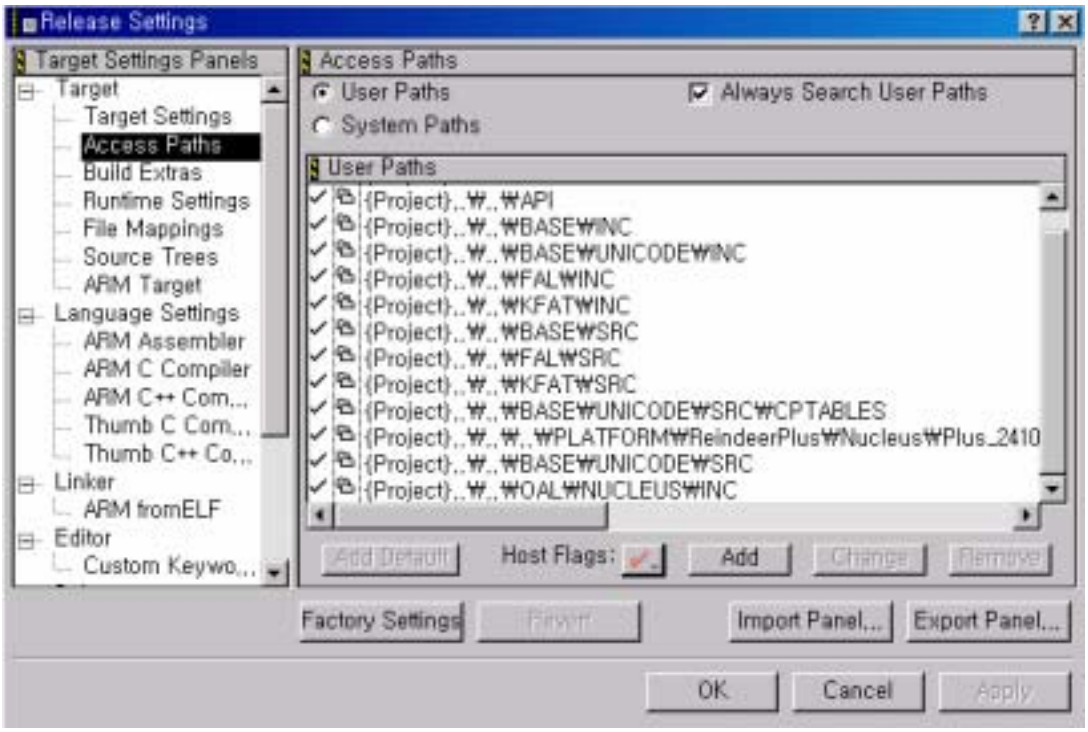
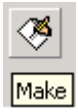


Figure 4-14. Set Access Paths

9. After setting the build options, press “OK” button to save.

10. Press 

button on ADS v1.2.

11. The project file build will start.

If the TFS4 building doesn’t encounter any compiling error, building TFS4 library is completed successfully.

4.2. TFS4 Porting to the Target OS

This section describes TFS4 porting process. The following picture shows the current step on the TFS4 porting process.

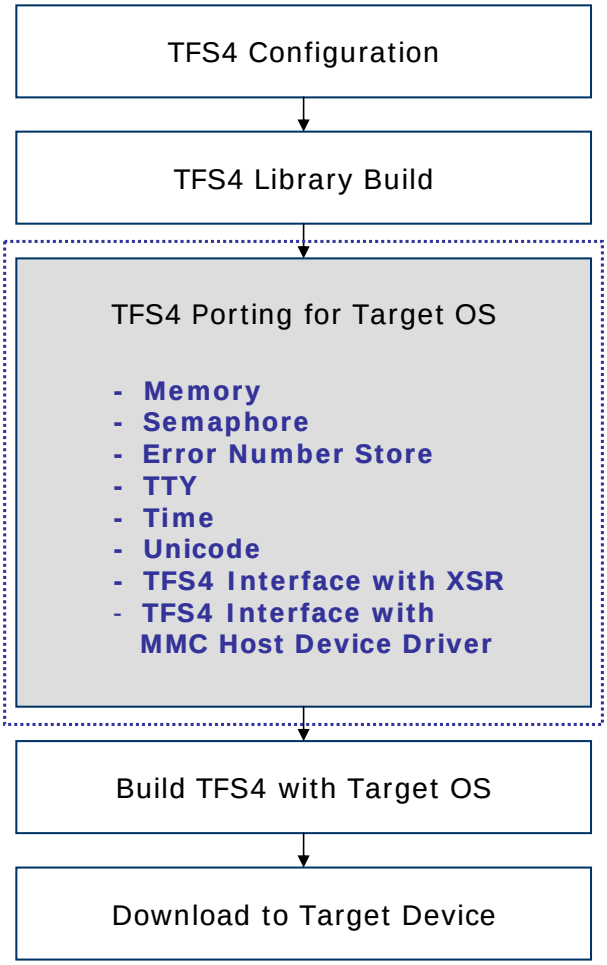


Figure 4-15. TFS4 Porting for Target OS

This document defines a porting sequence. XSR and MMC(or HSMMC) porting is done in advance, and TFS4 is ported to your target. TFS4 works with XSR and MMC(or HSMMC) together. Thus, if any error occurs, you cannot find whether error is from TFS4 itself or not. TFS4 cannot be tested alone.

XSR and MMC(or HSMMC) should be ported and tested before being integrated with TFS4. It is strongly recommend porting TFS4 after XSR and MMC(or HSMMC) are guaranteed to work reliably on target with no error.
In current TFS4 version, porting example source codes for Nucleus is supported.

The following figure shows configurable TFS4 source files. You have to configure them in

this section.

tfs4_config_base.h
tfs4_config_const.h
tfs4_config.h
tfs4_memory.h
tfs4_pdev_nand_xsr.h
tfs4_semaphore.h

tfs4_errno.c
tfs4_memory.c
tfs4_pdev_mmc_reindeer_plus.c
tfs4_pdev_nand_xsr.c
tfs4_semaphore.c
tfs4_time.c
tfs4_tty.c
tfs4_tuning.c

- Implementing part for target RTOS

1. Semaphore (tfs4_semaphore.c)
2. Memory allocation functions (tfs4_memory.c)
3. Error number store (tfs4_errno.c)
4. Time (tfs4_time.c)

- Implementing part for target device

1. MMC(or HSMMC) Host Driver (tfs4_pdev_mmc_reindeer_plus.c. This file name can be changed by user)
2. XSR (tfs4_pdev_nand_xsr.c)
3. UART print (tfs4_tty.c)

This document explains the TFS4 porting based on ReindeerPlus and Nucleus as a sample.

4.2.1. XSR Porting

For XSR porting, you can refer to XSR porting guide.
Here important thing is that you have to port XSR to your target and test it to verify reliability of it.

4.2.2. MMC(or HSMMC) Host Device Driver Development

MMC(or HSMMC) Host Device Driver is not implemented in TFS4. TFS4 user has to implement it if the target uses MMC(or HSMMC). TFS4 includes sample source code and APIs based on the SAMSUNG S3C2410S CPU (Based on ARM920T core) architecture.

This is the directory path of sample MMC(or HSMMC) host device driver.

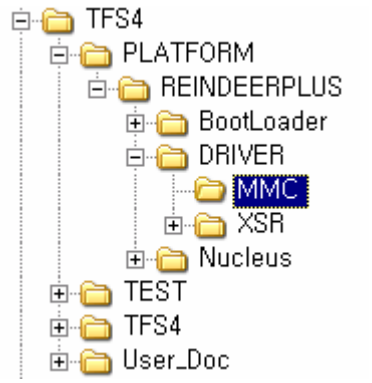


Figure 4-16. MMC(or HSMMC) Host Device Driver Path

In MMC(or HSMMC) directory, there are sample MMC(or HSMMC) host device driver source codes based on the S3C2410S CPU architecture, as follows.

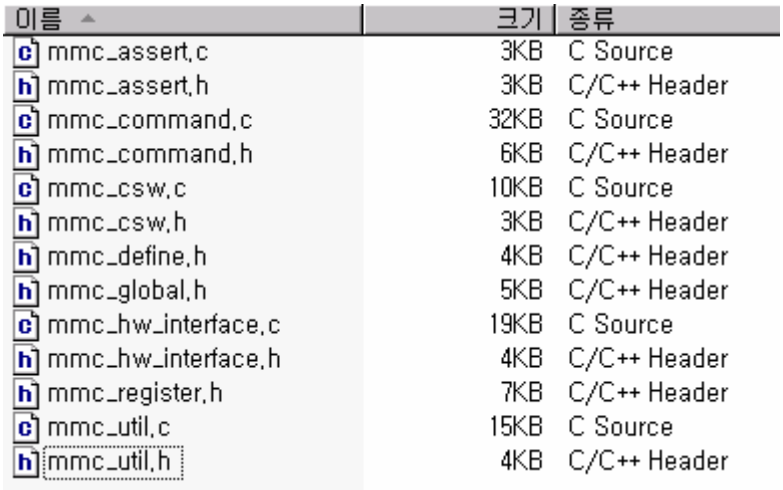


Figure 4-17. The Source File List of Sample MMC(or HSMMC) Host Device Driver

You can open them to see how they are implemented. You can write your MMC(or HSMMC) host device driver suitable to your target with the given sample MMC(or

HSMMC) APIs. The sample HSMMC APIs are similar to the sample MMC APIs. MMC Device Driver support only 1 bit bus transfer mode. If you want to use a 4 bit or 8 bit transfer mode, you have to use the HSMMC.
For its development, you can refer to Appendix "II. MMC (or HSMMC) Host Device Driver APIs."

The following table summarizes sample MMC(or HSMMC) APIs and features.

Table 6. MMC(or HSMMC) APIs

MMC APIs	Descriptions
mmc_init_driver	It initializes MMC(or HSMMC).
mmc_is_ready	It returns whether MMC(or HSMMC) initialization is fail or success.
mmc_read	It reads data per sector from MMC(or HSMMC).
mmc_write	It writes data per sector on MMC(or HSMMC).
mmc_get_stat	It retrieves the information of MMC(or HSMMC) device.

You can find the feature of MMC(or HSMMC) host device driver you have to implement, through the above listed APIs.

TFS4 file system requests read/write operation in sector (512 byte), physical information, etc. to MMC(or HSMMC) host device driver. You don't have to implement other features of MMC(or HSMMC) like lock/unlock, password, and force erase, because TFS4 file system does not use them.

The following is the data structures of MMC(or HSMMC) host device driver.

Table 7. Data Structure of MMC(or HSMMC) Host Device Driver

```
typedef struct {
t_uint32    uiDevSize;
t_uint32    uiSectorSize;
t_uint32    uiNumSectors;
t_uint8     bSectorsPerTrack;
t_uint8     bTracks;
t_uint16    wCylinders;
t_uint32    uiWPGroupSize;
t_uint32    uiWPStatus;
t_uint32    uiProductSN;
t_uint16    wOemID;
t_uint8     bManID;
t_uint8     bProductRev;
t_int8      chProductName[6];
t_uint8     bManDate;
t_uint8     bReserved;
} t_mmc_info;
```

The following table shows the data structure description of MMC(or HSMMC) host device driver.

Table 8. Data Structure Description of MMC(or HSMMC) Host Device Driver

MMC Data Structure	Description
uiDevSize	MMC(or HSMMC) device size in byte
uiSectorSize	Sector size in byte.
uiNumSectors	Number of sectors
bSectorPerTrack	Number of sectors per track
bTrack	Number of tracks. It means a head count in C/H/S conversion
wCylinders	Number of cylinders
uiWPGroupSize	Number of sectors in a write protection group
uiWPStatus	Write protection status
uiProductSN	Product serial number
wOemID	OEM/application ID
bManID	Manufacturer ID
bProductRev	Product revision number
chProductName[6];	Product name
bManDate	Manufacturing date. YYYYMMMM(b). year: YYYY(b) + 1997 month: MMMM(b)
bReserved	Reserved for future use

If TFS4 file system requires MMC(or HSMMC) information, MMC(or HSMMC) host device driver has to read the register value of MMC(or HSMMC) and pass them to TFS4 file system.

bSectorPerTrack, bTracks, and wCylinders are geometric values and do not exist in MMC(or HSMMC). They are calculated by using tfs4_pdev_get_geometrics() in the tfs4_pdev.c file. Refer to sample MMC(or HSMMC) host device driver.

The following shows the implemented MMC(or HSMMC) host device driver for ReindeerPlus; a device driver depends on target hardware.

Table 9. Sample MMC(or HSMMC) Host Device Driver for ReindeerPlus

```
t_int32     mmc_init_driver   (void)
t_int32     mmc_is_ready      (void)
t_int32     mmc_read    (t_uint8       *pBuf,       t_uint32
uiStartSector, t_uint32 uiNumSectors)
t_int32     mmc_write   (t_uint8       *pBuf,       t_uint32
uiStartSector, t_uint32 uiNumSectors)
void  mmc_get_stat      (t_mmc_info* pBuf);
```

You can write a MMC(or HSMMC) host device driver considering your target like the above sample. API is same, however the implemented source can be different depending on the target.

4.2.3. Bad Sector Manager

Now TFS4 v1.5 supports a new feature, bad sector manager that implements filter driver layer at the logical device driver level. When using magnetic disk as storage device, bad sectors can put the file system into trouble, such like long delay time during operation or catastrophic corruption. While the FTL that controls NAND Flash Memory conceals initial bad blocks from upper layer application, most magnetic disks don't handle bad sectors.

Bad Sector Manager is implemented as a filter driver. It intercepts all requests from the upper layer (FAL) and preprocesses them. When it receives a request (read or write), it checks whether the requested range of logical sector is on a bad sector, of which list are managed by bad sector manager. If then, bad sector manager may split and redirect the requests to the backup device.

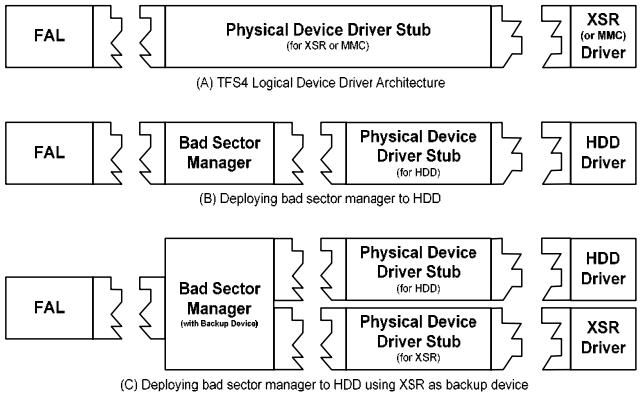


Figure 4-18 Deploying Bad Sector Manager

Following code snippet shows you an example of deploying bad sector manager to a hard-disk drive.

```
// Declaration
#define BSM_INIT
#include "tfs4_bsm.h"
DECLARE_DEVICE_CONTEXT(myhdd, "/dev/hdd");

// Physical Device Registration for myhdd1
tfs4_pdev_hdd_get_op(&stOp1);
tfs4_bsm_init_ex(PDEVICE_CONTEXT(myhdd), &stOp1, NULL, 0);
tfs4_bsm_get_op(PDEVICE_CONTEXT(myhdd), &stOp);
tfs4_pdev_reg(&stOp, TRUE, TRUE);
```

Following code snippet shows you another example of deploying bad sector manager to a hard-disk drive using XSR as a backup.

```
// Declaration
#define BSM_INIT
#include "tfs4_bsm.h"
DECLARE_DEVICE_CONTEXT(myhdd, "/dev/hdd");

// Physical Device Registration for myhdd2
tfs4_pdev_hdd_get_op(&stOp1);
tfs4_pdev_nand_xsr_get_op(&stOp2);
tfs4_bsm_init_ex(PDEVICE_CONTEXT(myhdd), &stOp1, &stOp2,
0xF0000);
```

```
tfs4_bsm_get_op(PDEVICE_CONTEXT(myhdd), &stOp);
tfs4_pdev_reg(&stOp, TRUE, TRUE);
```

Table 10 Bad Sector Manager APIs and Macro

BSM APIs and Macro	Descriptions
`tfs4_bsm_init_ex`	Initialize BSM device and specify main device and backup device.
`tfs4_bsm_get_op`	Get a physical_device_op structure of BSM device.
`tfs4_bsm_format`	Build initial data structures of BSM device.
`tfs4_bsm_reallocate`	Add alternate translation rule to the BSM device.
`DECLARE_DEVICE_CONTEXT`	Create an instance of BSM device
`PDEVICE_CONTEXT`	Returns a pointer to an instance of BSM device

4.2.4. Common IOCTLs

Lower hardware driver such like XSR, MMC, or other devices may handle additional IOCTLs for supporting device-specific features. Currently, following IOCTLs are defined. Implementation of devices that does not support these features may ignore the requests.

enuIOCTL_GET_LASTERROR

When requested, IOCTL function should return the length of sectors successfully written or read at the previous write or read request. For example, if an error occurred while writing 55th sector at the previous write request, pfIOCTL of the driver implementation should return 54 as its return value.

4.2.5. TFS4 Porting

This section describes how TFS4 is ported to your target. Mostly, what you have to do for TFS4 porting in this section is writing the source codes related with your target and RTOS. Then, you have to define it in the header file.

This is a source file list to configure from the TFS4 source files.

Table 11. TFS4 Source Files Being Ported to Target

Dependency	Porting Parts	Source files to configure	Header file to define
Target RTOS	Memory configuration	tfs4_memory.c	tfs4_config.h
	Semaphore configuration	tfs4_semaphore.c	tfs4_semaphore.h
	Error Number	tfs4_errno.c	tfs4_errno.h
Target Device	TTY configuration	tfs4_tty.c	tfs4_tty.h
	Time configuration	tfs4_time.c	tfs4_time.h
	Unicode configuration	tfs4_unicode_xxx.c	tfs4_unicode_char.h
	TFS4 Interface with XSR	tfs4_pdev_nand_xsr.c	tfs4_pdev_nand_xsr.h
	TFS4 Interface with MMC(or HSMMC) host device driver	tfs4_pdev_mmc_reindeer_plus.c	tfs4_pdev_mmc_reindeer_plus.ch

You can find almost all of the files in the directory
“C:\TFS4\TFS4\OAL\NUCLEUS\SRC”

4.2.5.1. Memory configuration

There are two types of memory allocation from OS:

- Memory pool: Nucleus, RTKE
- Plain memory: pSOS, Linux

Memory pool is a preoccupied memory region for use. TFS4 memory pool can be created when OS or TFS4 is initialized. If the memory pool is created when TFS4 initialization, you have to write a `tfs4_memory_init()` function.

If your target OS has a plain memory type, target OS dynamically allocates a TFS4 memory. In that case, you don't have to do nothing in `tfs4_memory_init()`; it always returns 0. Set `bIsMemoryInitialized` as true. If memory initialization is success, then it returns 0, success.

This section shows TFS4 memory configuration with a sample code, which is implemented based on Nucleus.

<TFS4 memory configuration for Nucleus>

In the provided TFS4 source files, there is an implemented source for memory allocation as a sample. It is implemented for Nucleus.

1. Execute an ADS 1.2, a build tool, on your host.

2. Open a tfs4_memory.c. The file directory path is C:\TFS4\TFS4\OAL\NUCLEUS\SRC.

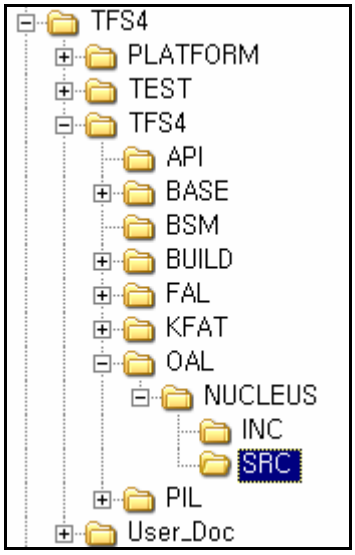


Figure 4-19. Directory Path of tfs4_memory.c

3. The tfs4_memory.c file is opened on ADS 1.2 editor as follows.

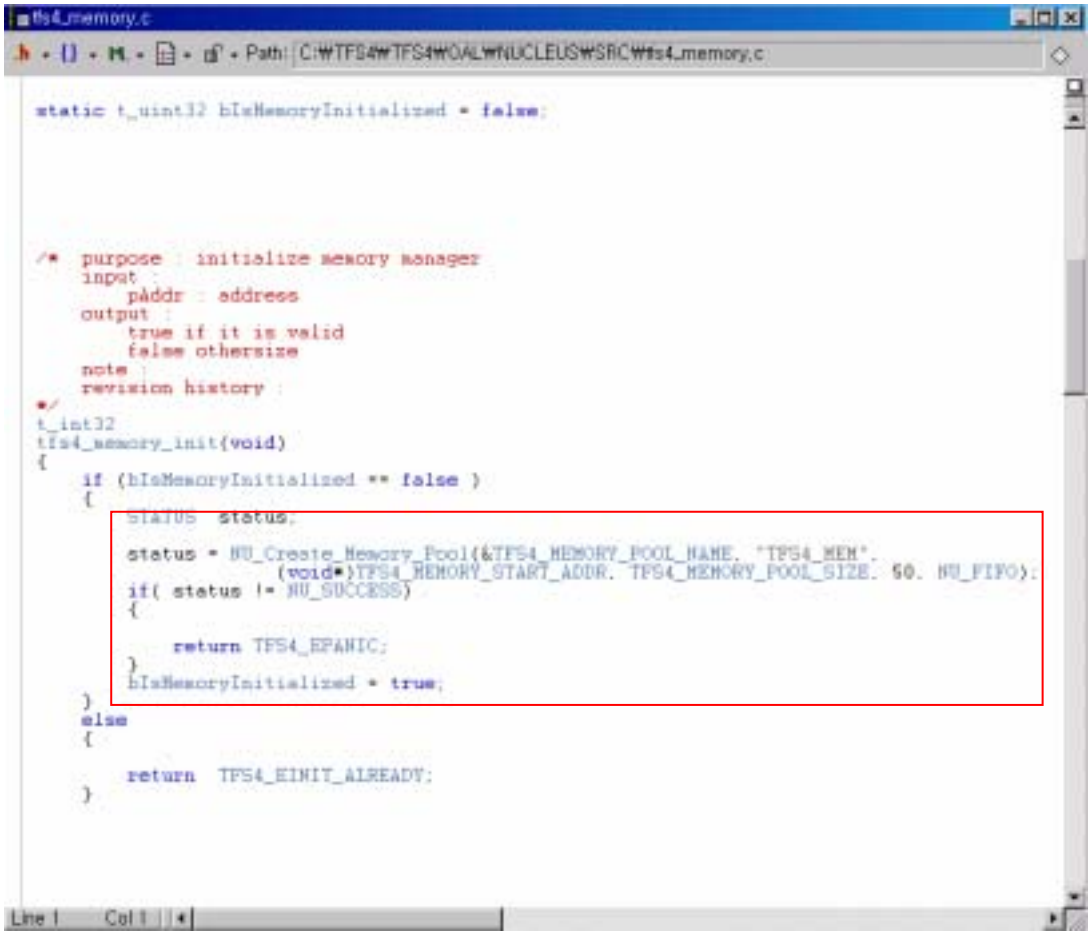


Figure 4-20. tfs4_memory.c

For Nucleus, a memory pool is used for memory allocation. When TFS4 is initialized, the memory is allocated from the memory pool.

You have to add a function for the memory pool creation function in the `tfs4_memory_init()` as the above. The code for creating the Nucleus memory pool, `NU_Create_Memory_Pool`, is already implemented as a sample.

4. This is an implementation guideline according to target OS and memory allocation type.

Table 12. Memory-related Implementation Guideline

OS	Memory allocation type		Implementation
Nucleus, RTKE	Memory pool	OS creates a memory pool	`tfs4_memory_init()` returns 0, because OS already created a memory pool. You have to define the TFS4 memory pool name in tfs4_memory.h as follows. `#define TFS4_MEMORY_POOL_NAME TFS4_Memory` It is to notify a memory pool name to TFS4
		TFS4 creates a memory pool	You implement a function for memory pool creation. Currently, the memory pool creation function for Nucleus is implemented in tfs4_memory.c/h.
pSOS, Linux	Plain memory		OS dynamically allocates a TFS4 memory. You don't have to do nothing in `tfs4_memory_init()`; it always returns 0.

According to your target RTOS, you have to make TFS4 use a memory; it is whether a memory pool is used or not.

5. This is the implemented sample source for using a memory pool in tfs4_memory.c. Those are developed on Nucleus.

Table 13. Implemented Memory-related Sources on Nucleus

```
#include <stdio.h>
#include <ctype.h>

#include "tfs4_types.h"
#include "tfs4_memory.h"
#include "tfs4_debug.h"
#include "tfs4_errno.h"
#include "tfs4_oal.h"
```

```
NU_MEMORY_POOL  TFS4_Memory;

static t_uint32 bIsMemoryInitialized = false;

/*  purpose : initialize memory manager
    input :
       pAddr : address
    output :
       true if it is valid
       false otherwise
    note :
    revision history :
*/
t_int32
tfs4_memory_init(void)
{
    if (bIsMemoryInitialized == false )
    {
       STATUS  status;

       status = NU_Create_Memory_Pool(&TFS4_MEMORY_POOL_NAME,
"TFS4_MEM",
                  (void*)TFS4_MEMORY_START_ADDR,
TFS4_MEMORY_POOL_SIZE, 50, NU_FIFO);
       if( status != NU_SUCCESS)
       {

          return TFS4_EPANIC;
       }
       bIsMemoryInitialized = true;
    }
    else
    {

       return  TFS4_EINIT_ALREADY;
    }
    return 0;
}

/*  purpose : reset memory manager
    input :
       none
    output :
       0 on success
       < 0 on failure
    note :
    revision history :
*/
t_int32
tfs4_memory_reset(void)
{
    //// add memory manager reset code here
    STATUS  status;

    if (bIsMemoryInitialized == true )
```

```
    {
        status                                                  =
NU_Delete_Memory_Pool(&TFS4_MEMORY_POOL_NAME);
        if( status != NU_SUCCESS)
        {

            return TFS4_EPANIC;
        }
    }
    bIsMemoryInitialized = false;

    return 0;
}

/*  purpose : check if the given memory address is valid
    input :
        pAddr : address
    output :
        true if it is valid
        false otherwise
    note :
    revision history :
*/
t_uint32
tfs4_is_valid_addr(void *pAddr, t_uint32 dwSize)
{
    t_uint32 dwAddr;

    dwAddr = (t_uint32) pAddr;

    if (pAddr != NULL)
    {
        if ((dwAddr >= TFS4_MEMORY_START_ADDR) &&
            (dwAddr  +  dwSize)  <=  (TFS4_MEMORY_START_ADDR  +
TFS4_MEMORY_POOL_SIZE))
        {
            return true;
        }
        else
        {

            return false;
        }
    }

     return false;
}


/*  purpose : allocation memory
    input :
        nSize : allocation size
    output :
        0 > : Success and available pointer
        0   : Memory allocation fail
```

```
    note :
    revision history :
        26-MAR-2004 [DongYoung Seo]: First Writing
*/
void *
tfs4_memory_alloc(t_uint32 nSize)
{
    void *pMem;

    if(   NU_Allocate_Memory(&TFS4_MEMORY_POOL_NAME,   &pMem,
nSize, NU_NO_SUSPEND) != NU_SUCCESS )
    {
        return  NULL;
    }
     return pMem;
}

/*  purpose : deallocate memory
    input :
        pMem : pointer of memory
    output :
        none
    note :
    revision history :
        26-MAR-2004 [DongYoung Seo]: First Writing
*/
void
tfs4_memory_free(void *pMem)
{
    NU_Deallocate_Memory(pMem);
 }
```

The following explains the above sample source codes of being ported to Nucleus.

Table 14. Description of Sample Source Codes

Function	Description
`tfs4_memory_init`	TFS4 initializes the memory pool as follows: - If OS creates a TFS4 memory pool, `tfs4_memory_init()` returns 0. - If TFS4 creates its own memory pool, `tfs4_memory_init()` creates the TFS4 memory pool. `tfs4_memory_init()` returns 0 on success, and returns -1 on failure from the implemented function. To confirm if `tfs4_memory_init()` is successfully performed, `bIsMemoryInitialized` should be set as true.

`tfs4_memory_reset`	TFS4 resets the created memory pool as follows: - If OS creates a TFS4 memory pool, `tfs4_memory_reset()` returns 0 on success. It does not mean the memory pool is deleted, because it is created by OS. - If TFS4 creates its own memory, `tfs4_memory_reset()` deletes the TFS4 memory pool. `tfs4_memory_reset()` returns 0 on success and `bIsMemoryInitialized` should be set as false. If the memory pool is not removed, `tfs4_memory_reset` returns -1 from the implemented function and `bIsMemoryInitialized` is not changed.
`tfs4_memory_alloc`	It functions same as `malloc()` of standard library. It is called while tfs4 is running. It returns the starting address of the allocated memory on success, and it returns `NULL` on failure.
`tfs4_memory_free`	It functions same as `free()` of standard library. It is called while tfs4 is running. It releases the allocated memory.

In order to implement a memory pool creation function suitable for your target OS, you have to modify the internal function of the `tfs4_memory_init()`, `tfs4_memory_reset()`, `tfs4_memory_alloc()`, and `tfs4_memory_free()`.

Currently, the memory pool related functions for Nucleus are implemented in tfs4_memory.c.

6. If your target OS is Nucleus and you creates a memory pool, you have to define the TFS4 memory pool name in tfs4_memory.h as follows.

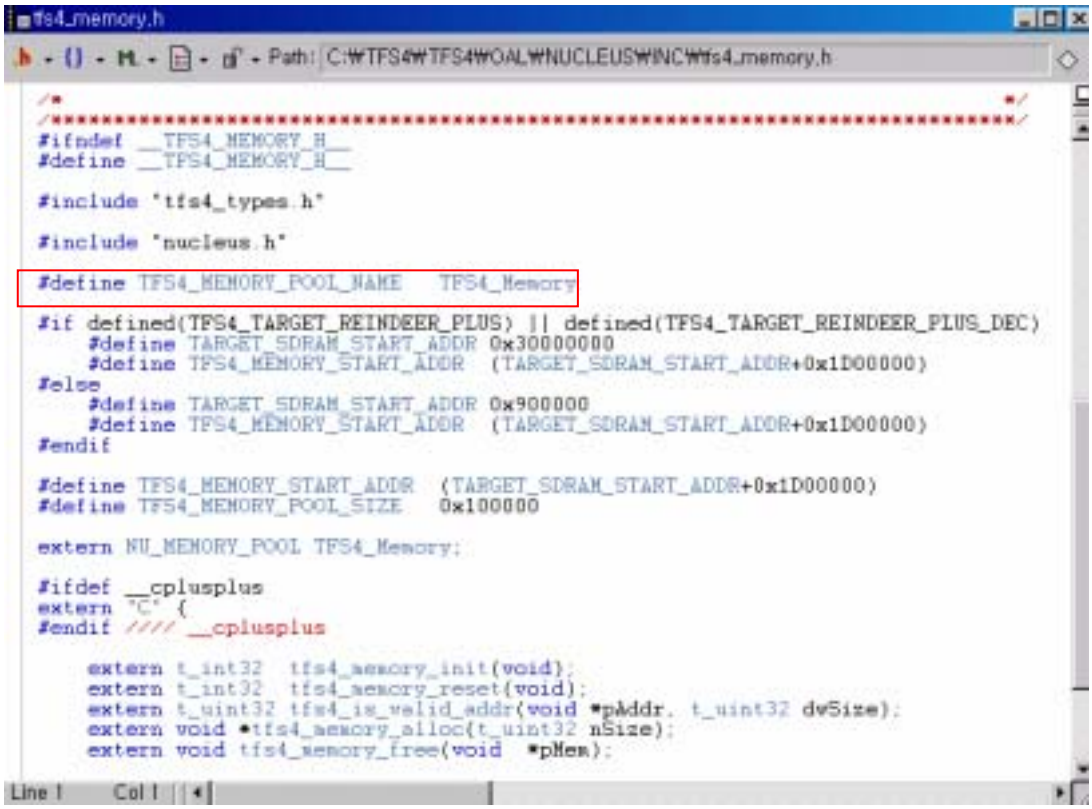


Figure 4-21. Define a Memory Pool Name in tfs4_memory.h

It is to prevent errors from where allocation and free function is implemented.

7. If you implement a source code for memory allocation of target OS, define them in tfs4_config_base.h file as follows.

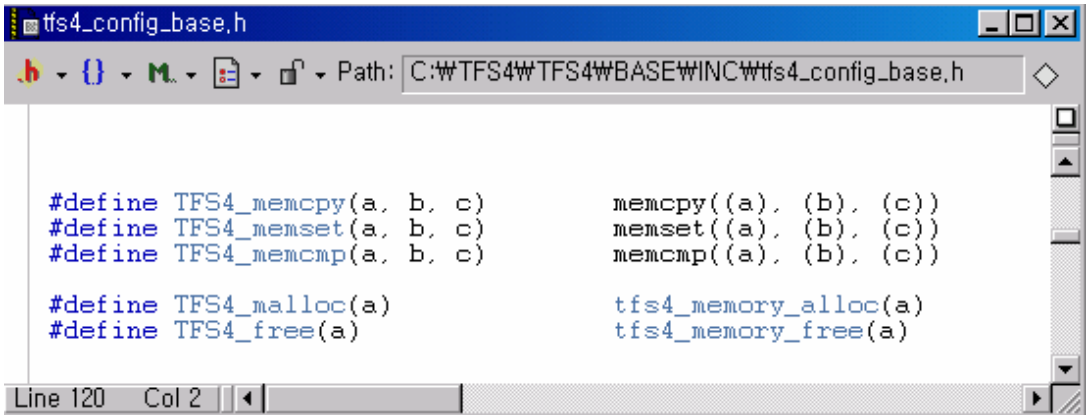


Figure 4-22. Define a tfs4_memory_alloc in tfs4_config_const.h

In the current version, `memcpy`, `memset`, and `memcmp` use a standard library. Modify tfs4_config_const.h to use the user-created function, instead of compiler-supported library. If they are not running, you need to implement them such as `TFS4_malloc` or

`TFS4_free`.

Now you've done the TFS4 memory configuration.

Reference

- TARGET_SDRAM_START_ADDR and TARGET_SDRAM_START_ADDR is physical address of SDRAM.

- TFS4_MEMORY_POOL_NAME, TFS4_MEMORY_START_ADDR, and TFS4_MEMORY_POOL_SIZE are added for Nucleus RTOS in tfs4_memory.h.
For Nucleus, the values are necessary for creating a memory pool. But they are not needed for pSOS or Linux.

- `TFS4_MEMORY_START_ADDR` and `TFS4_MEMORY_POOL_SIZE` should be set, with reference to a memory map; the memory map is specified by OS porting policy. It is to prevent overlapping with another pool.

For Nucleus, a memory pool created by `tfs4_memory_init()` is shared with XSR. Thus, the memory pool size should be set enough.

Note

- We plan to modify XSR to use its own memory pool, not shared with TFS4 memory pool.

- For more information of memory pool size, refer to the memory usage of TFS4 in 2.2 Target.

4.2.5.2. Semaphore configuration

Semaphore is to prevent other users from opening the same file or directory at that time when a file or directory is opened. It makes it possible to keep a same file or directory open but do not read/write access at same time. Its purpose is to preserve the integrity of data while you are using it.

TFS4 uses a semaphore when a file or directory is accessed. But the TFS4 semaphore can be created by OS or TFS4 at different time, according to OS. It can be created when:

- ☐ OS initialization

Semaphore is already created on memory when OS porting. TFS4 uses the created semaphore. When TFS4 needs a semaphore, the created semaphore address is returned. For that case, you have to implement a `tfs4_sm_p()` and `tfs4_sm_v()` to obtain and release the OS-created semaphore.

- ☐ TFS4 initialization, termination

Semaphore is dynamically created when a `tfs4_init()` is executed. You have to implement `tfs4_sm_create()`, `tfs4_sm_delete()`, `tfs4_sm_p()`, and

`tfs4_sm_v()`. They should be suitable for target OS. After TFS4 creates a semaphore by `tfs4_sm_create()`, semaphore is actually retrieved by `tfs4_sm_p()` when a file or directory is accessed. For Nucleus and pSOS, that rule is applied.

TFS4 uses two types of semaphores:

- Directory semaphore
- File semaphore

When TFS4 tries to create a semaphore, the semaphore type is checked by the semaphore name as a parameter. Only one semaphore is created when a semaphore is called.

<TFS4 semaphore configuration for Nucleus>

In the provided TFS4 source files, there is an implemented source for semaphore creation as a sample. It is implemented for Nucleus.

1. Execute an ADS 1.2, a build tool, on your host.

2. Open a tfs4_semaphore.c. The file directory path is
“C:\TFS4\TFS4\OAL\NUCLEUS\SRC”.

3. The tfs4_semaphore.c file is opened on ADS 1.2 editor as follows.

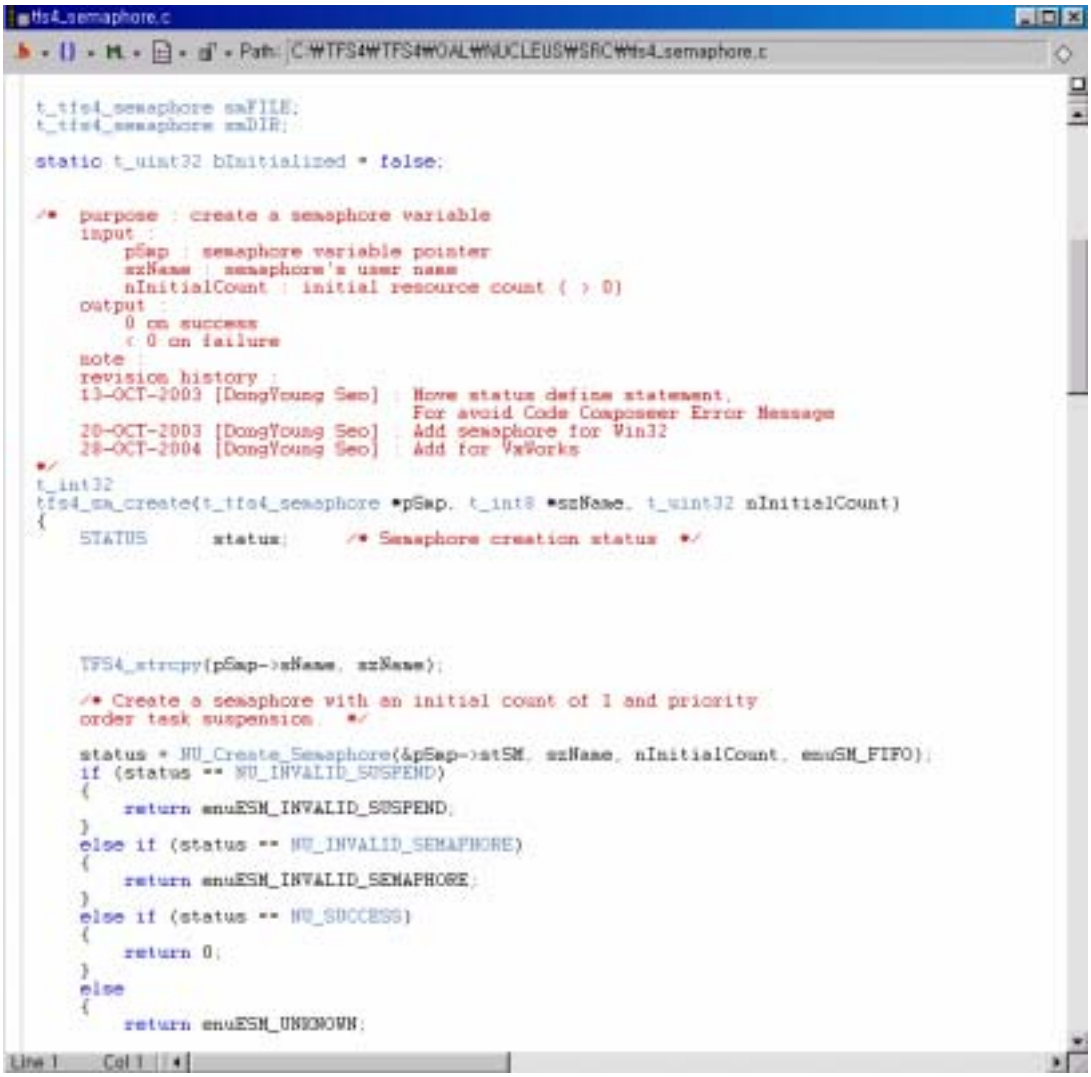


Figure 4-23. tfs4_semaphore.c

4. Semaphore configuration depends on target OS. This is the semaphore implementation guideline depending on target OS.

Table 15. Semaphore Implementation Guideline

OS	Semaphore Creation Type	Implementation
RTKE	OS creates a semaphore	Configure the semaphore related functions in the tfs4_semaphore.c/h file
Nucleus, pSOS	TFS4 creates a semaphore	Implement the semaphore related functions such as the following functions in the tfs4_semaphore.c/h file:

		- Create: `tfs4_sm_create()` - Delete: `tfs4_sm_delete()` - Obtain: `tfs4_sm_p()` - Release: `tfs4_sm_v()` They should be implemented for your OS.

5. You can find the implemented source based on Nucleus, pSOS, and RTKE in the tfs4_semaphore.c/h file. You should define the OS in the tfs4_config_const.h file for using the implemented source.

If you use different target OS like Linux, you should add the semaphore related source code suitable for the OS.

<For implementing the semaphore functions on another OS>

6. Define the OS-defined variable type with `t_semaphore` in the tfs4_semaphore.h file as follows.

Table 16. type definition OS-Defined Variable

```
#if (TFS4_OS == TFS4_NUCLEUS)
typedef NU_SEMAPHORE    t_semaphore;      //// for Nucleus
#elif (TFS4_OS == TFS4_WIN32)
typedef HANDLE    t_semaphore;      //// for Windows
#elif (TFS4_OS == TFS4_PSOS)
typedef unsigned long   t_semaphore; //// for pSOS
#elif (TFS4_OS == TFS4_RTKE)
typedef     unsigned char     t_semaphore;      //// for RTKE
#else
typedef t_uint32       t_semaphore;  //// for Others
#endif
```

It is to change the OS-defined variable type to commonly used one for TFS4, `t_semaphore`, when semaphore sources are compiled.

7. Each OS has different type of semaphore structure. Thus, the semaphore structure of tfs4_semaphore.h has to be modified, according to OS.

The following shows the OS-defined arguments of semaphore functions.

Table 17. Typedef OS-Defined Argument

```
typedef enum {
   enuSM_PRIOR     = NU_PRIORITY,
```

```
    enuSM_FIFO     = NU_FIFO,
    enuSM_SUSPEND   = (t_int32)NU_SUSPEND,
    enuSM_NO_SUSPEND = NU_NO_SUSPEND
} t_sm;
```

It is to change the OS-defined structure to commonly used one for TFS4 when semaphore sources are compiled.

8. The following describes the semaphore functions implemented in the tfs4_semaphore.c file. You can use them if your target OS is Nucleus, pSOS, and RTKE.

If not, you can newly implement properly them according to your target OS, by referring to semaphore functions your OS provides, in the tfs4_semaphore.c file.

- tfs4_sm_create()

To use a semaphore for TFS4, the semaphore should be created first. TFS4 semaphore can be created when OS or TFS4 is initialized according to OS.

Here is the `tfs4_sm_create()` implemented for Nucleus; Nucleus use the TFS4-created semaphore. You can just use the implemented source code by defining the OS in the tfs4_config_const.h file.

```
t_int32
tfs4_sm_create(t_tfs4_semaphore *pSmp, t_int8 *szName, t_uint32 nInitialCount)
{
    STATUS          status;        /* Semaphore creation status    */

    TFS4_strcpy(pSmp->sName, szName);

    /* Create a semaphore with an initial count of 1 and priority
    order task suspension.    */

    status    =    NU_Create_Semaphore(&pSmp->stSM,    szName,    nInitialCount,
enuSM_FIFO);
    if (status == NU_INVALID_SUSPEND)
    {
        return enuESM_INVALID_SUSPEND;
    }
    else if (status == NU_INVALID_SEMAPHORE)
    {
        return enuESM_INVALID_SEMAPHORE;
    }
    else if (status == NU_SUCCESS)
    {
        return 0;
    }
    else
    {
        return enuESM_UNKNOWN;
    }
}
```

It depends on OS. If OS already created a semaphore, `tfs4_sm_create()` returns 0.

TFS4 needs two semaphores; a file semaphore and directory semaphore. They can be checked by `szName` as a `tfs4_sm_create()` parameter. It returns 0 on success and an error on failure; the returned error is defined in `t_sm_error` structure in the tfs4_semaphore.h. file. The file semaphore name is "`TFIL`," and the directory semaphore name is "`TDIR`."

- tfs4_sm_delete()

This function deletes a semaphore. You have to implement `tfs4_sm_delete()` according to the target OS.

Here is the `tfs4_sm_delete()` implemented for Nucleus, pSOS, and RTKE in the tfs4_semaphore.c file.

```
t_int32
tfs4_sm_delete(t_tfs4_semaphore *pSmp)
{
    STATUS     status;



    status = NU_Delete_Semaphore(&pSmp->stSM);
    if (status != NU_SUCCESS)
    {
        return enuESM_UNKNOWN;
    }

    return 0;
}
```

It returns 0 on success and an error on failure; the returned error is defined by the `t_sm_error` structure in the tfs4_semaphore.h.file. You can use the implemented source code just by defining the OS in the tfs4_config_const.h file.

- tfs4_sm_p()

This function obtains the created semaphore. You have to implement `tfs4_sm_p()` for target OS.

Here is the `tfs4_sm_p()` implemented for Nucleus, pSOS, and RTKE

```
t_int32
tfs4_sm_p(t_tfs4_semaphore *pSmp)
{
    STATUS     status;
```

```
    status      =           NU_Obtain_Semaphore(&pSmp->stSM,
(t_uint32)enuSM_SUSPEND);

    if (status != NU_SUCCESS)
    {
        return enuESM_UNKNOWN;
    }

    return 0;
}
```

It returns 0 on success and an error on failure; the returned error is defined by the `t_sm_error` structure in the tfs4_semaphore.h.file. You can just use the implemented source code by defining the OS in the tfs4_config_const.h file.

- tfs4_sm_v()

This function releases the semaphore. You have to implement `tfs4_sm_v()` for target OS.

Here is a sample implementation of `tfs4_sm_v()` for Nucleus, pSOS, and RTKE.

```
t_int32
tfs4_sm_v(t_tfs4_semaphore *pSmp)
{
    STATUS     status;



    status =  NU_Release_Semaphore(&pSmp->stSM);

    if (status != NU_SUCCESS)
    {
        return enuESM_UNKNOWN;
    }

    return 0;
}
```

It returns 0 on success and an error on failure; the returned error is defined by the `t_sm_error` structure in the tfs4_semaphore.h. You can just use the implemented source code by defining the OS in the tfs4_config_const.h file.

4.2.5.3. Error Number store

Error number (errno) has to be stored on memory to return an error number when directory or file operation occur the error.

Since Linux or Windows is running on its own memory, the error number of a process is not changed by another process even if the error number is defined as global variable.

But RTOS like Nucleus or pSOS shares memory. A global variable can be accessed by any task on RTOS. It happens that the error number of a task can be changed by another task. For that reason, error number has to be stored on separate task; Nucleus does it. But, for pSOS it is not implemented yet.

Nucleus has a reserved region inside TCB (Task Control Block) that stores the task information including the errno. pSOS is designed to store the task information on global region. If you use another OS, you have to specify a region for storing an error number depending on your OS.

Storing the error information, errno, is composed of two functions:

- tfs4_err_set_errno()
- tfs4_err_get_errno()

`tfs4_err_set_errno()` sets an error number to the errno, which is retrieved as an integer parameter, on separate task region. `tfs4_err_get_errno()` returns an errno from the separate task region.

By implementing the above functions according to your target OS, you can check the latest error and what it is.

This section shows TFS4 error number-store configuration with a sample code, which is implemented based on Nucleus.

<TFS4 error number-store configuration for Nucleus>

1. Execute an ADS 1.2, a build tool, on your host.

2. Open a tfs4_errno.c. The file directory path is “C:\TFS4\TFS4\OAL\NUCLEUS\SRC.”

3. The tfs4_errono.c file is opened on ADS 1.2 editor as follows.

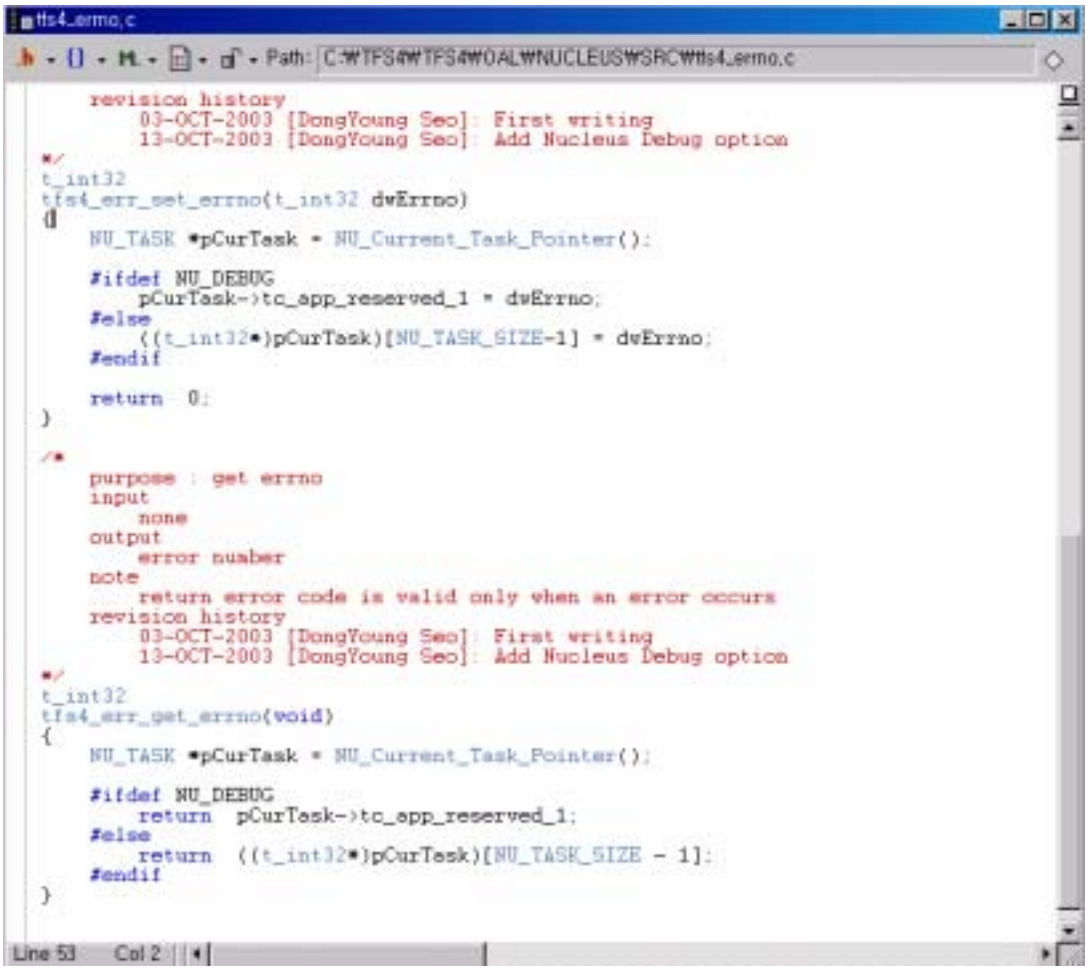


Figure 4-24. tfs4_errno.c

In case that your target OS is Nucleus.
But if you use other target OS, you have to implement the above two functions according to the OS.

4.2.5.4. TTY configuration

TTY is the most widely used type of emulation for PC computer communications.

In TFS4, TTY has to be configured for using a test shell. It is to get a debugging message from target while TFS4 or XSR is running on target, through UART. You can configure it at tfs4_tty.c file according to your target; functions are already implemented in tfs4_tty.c/h.

< TTY configuration of TFS4 >

1. Execute an ADS 1.2, a build tool, on your host.

2. Open a tfs4_tty.c. The file directory path is ”C:\TFS4\TFS4\OAL\NUCLEUS\SRC”.

3. The tfs4_tty.c file is opened on ADS 1.2 editor as follows.

Figure 4-25. tfs4_tty.c

The following represents the syntaxes of the implemented source codes based on the UART device driver of ReindeerPlus to get or print a text from target.

```
t_int32 tfs4_get_char(void)
t_int32 tfs4_get_int(void)
t_int32 tfs4_gets(t_int8 *pBuff)
void tfs4_printf(const t_int8 *fmt,...);
```

4. You have to configure them appropriately to your target in order to use UART device.

Notice

UART is to print the debugging information for TFS4 while TFS4 is tested. Input code is implemented in the tfs4_tty.c file. The input functions are `tfs4_get_char()`, `tfs4_get_int()`, and `tfs4_gets()`.

4.2.5.5. Time configuration

Time configuration needs to be done for TFS4 to get a specific time when file or directory is created, accessed, or written.

It depends on OS or compiler. It is implemented in tfs4_time.c and tfs4_config_base.h.

This section shows TFS4 time configuration with a sample code, which is implemented based on Nucleus.

<TFS4 time configuration for Nucleus>

1. Execute an ADS 1.2, a build tool, on your host.

2. Open a tfs4_time.c. The file directory path is ”C:\TFS4\TFS4\OAL\NUCLEUS\SRC”.

3. The tfs4_time.c file is opened on ADS 1.2 editor as follows.

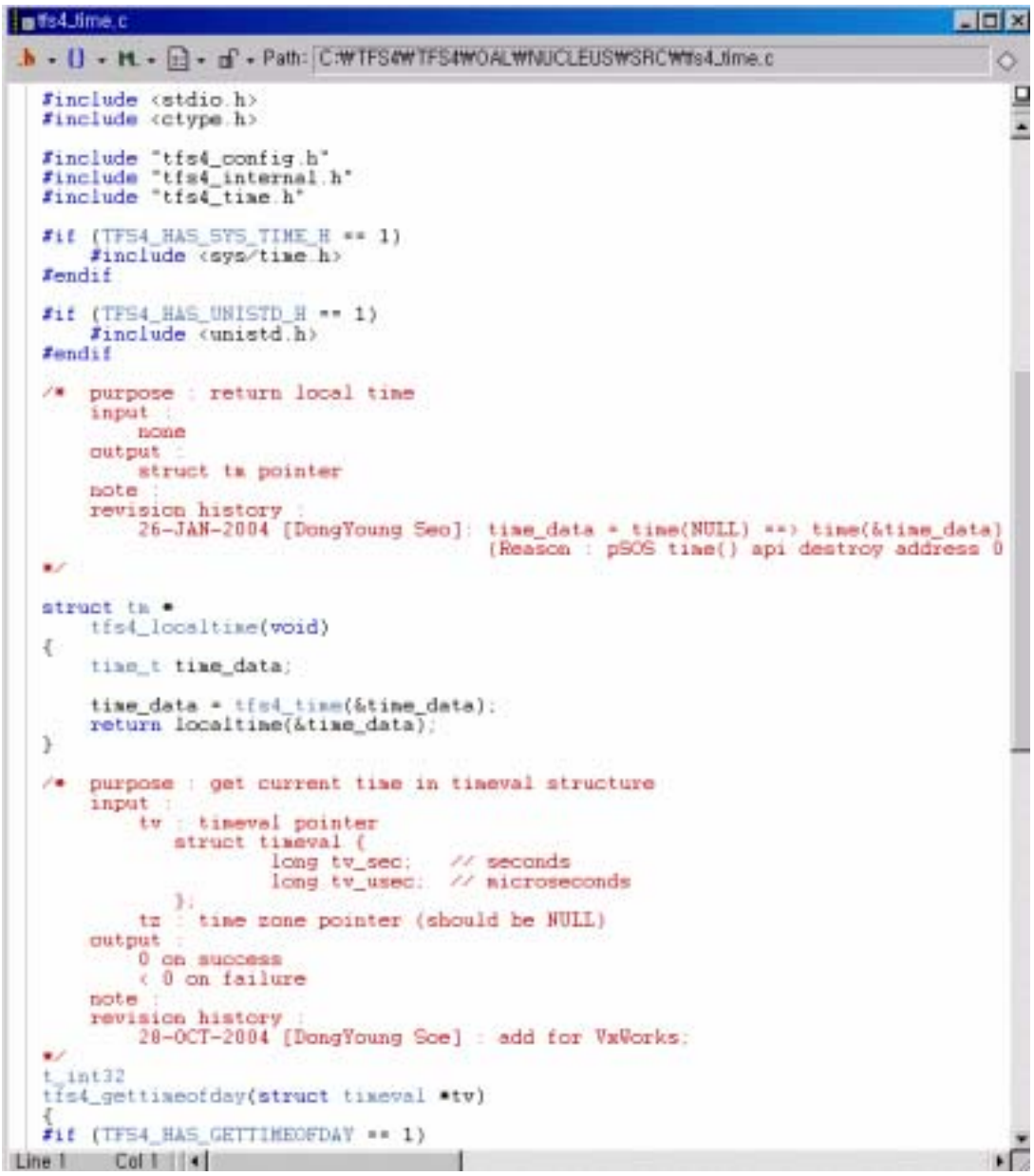


Figure 4-26. tfs4_time.c

The following represents the syntaxes of the source codes implemented to get or print a text from target.

```
struct tm *tfs4_localtime(void)
t_int32 tfs4_gettimeofday(struct timeval *tv)
void tfs4_get_time(t_uint16 *pwDate, t_uint16 *pwTime, t_uint8
*pbMSec)
```

You have to configure them appropriately to your target OS or compiler.

- `tfs4_localtime()` returns a local time by storing the current time in a `struct tm`.
- `tfs4_gettimeofday()` returns the current time by storing it in a `struct timeval`. It is called in `tfs4_get_time()`. If `tfs4_localtime()` is composed, the `tfs4_localtime()` is called.
- `tfs4_get_time()` stores a date and time separately as described in the `tfs4_stat()` of TFS4 Programmer's Guide.

Reference

<Date and Time Formats>
Many FAT file systems do not support Date/Time other than `DIR_WrtTime` and `DIR_WrtDate`. For this reason, `DIR_CrtTimeMil`, `DIR_CrtTime`, `DIR_CrtDate`, and `DIR_LstAccDate` are actually optional fields. `DIR_WrtTime` and `DIR_WrtDate` must be supported, however. If the other date and time fields are not supported, they should be set to 0 on file create and ignored on other file operations.

<Date Format>
A FAT directory entry date stamp is a 16-bit field that is basically a date relative to the MS-DOS epoch of 01/01/1980. Here is the format (bit 0 is the LSB of the 16-bit word, bit 15 is the MSB of the 16-bit word):

- Bits 0.4: Day of month, valid value range 1-31 inclusive.
- Bits 5.8: Month of year, 1 = January, valid value range 1.12 inclusive.
- Bits 9.15: Count of years from 1980, valid value range 0.127 inclusive (1980.2107).

<Time Format>
A FAT directory entry time stamp is a 16-bit field that has a granularity of 2 seconds. Here is the format (bit 0 is the LSB of the 16-bit word, bit 15 is the MSB of the 16-bit word).

- Bits 0.4: 2-second count, valid value range 0.29 inclusive (0 . 58 seconds).
- Bits 5.10: Minutes, valid value range 0.59 inclusive.
- Bits 11.15: Hours, valid value range 0.23 inclusive.

The valid time range is from Midnight 00:00:00 to 23:59:58.

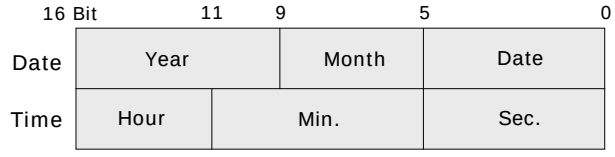


Figure 4-27. Data Format of Date and Time

4.2.5.6. TFS4 Interface with XSR

This section describes the XSR configuration. It is for TFS4 to use XSR, not for XSR configuration itself.

TFS4 uses XSR to perform several operations to Physical NAND device. Thus, you have to configure the NAND device setting.

XSR v1.4.0 provides multi-partition for file system. It make several file system device on a physical NAND. TFS4 also support XSR multi-partition feature.

This is a sample codes to use XSR v1.4.0. It assumes two partitions for file system on BM. The partition information is like below
We assume that the NAND Device has 1024 blocks.

file system partition 0 (block 0 ~ block 799)	file system partition 1

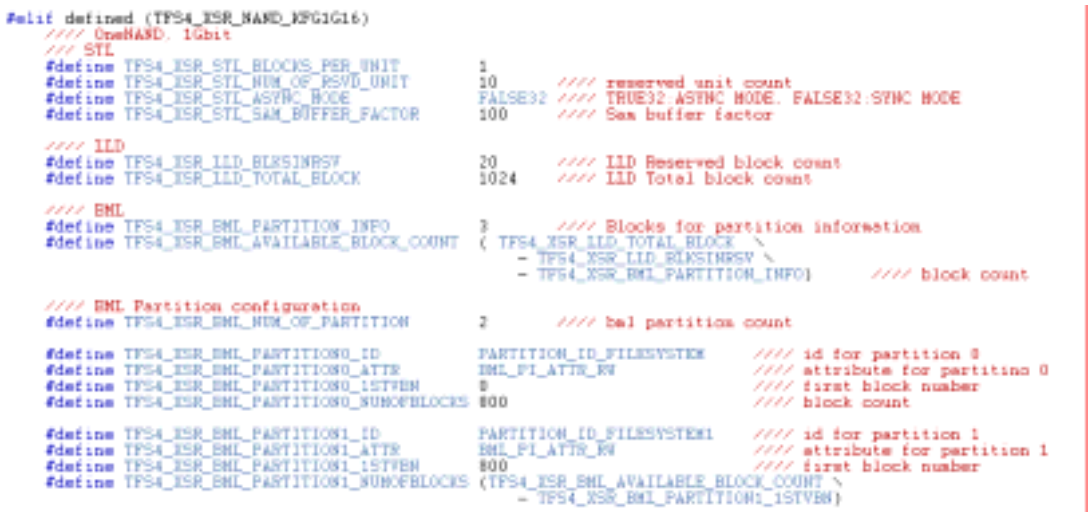
1. Execute an ADS 1.2, a build tool, on your host.

2. Open a tfs4_pdev_nand_xsr.h. The file directory path is ”C:\TFS4\SRC\TFS4\PIL\INC”.

3. TFS4_pdev_nand_xsr.h includes the all XSR-related settings as below.

```
#define TFS4_XSR_BML_PARTITION_INFO     4   //// Blocks for partition information for XSR v1.5 RC5

//// OneNAND, 1Gbit
/// STL
#define TFS4_XSR_STL_BLOCKS_PER_UNIT        1
#define TFS4_XSR_STL_NUM_OF_RSVD_UNIT       10      //// reserved unit count
#define TFS4_XSR_STL_ASYNC_MODE             FALSE32 //// TRUE32:ASYNC MODE, FALSE32:SYNC MODE
#define TFS4_XSR_STL_SAM_BUFFER_FACTOR      100     //// Sam buffer factor

//// LLD
#define TFS4_XSR_LLD_RLRSINRSV              20      //// LLD Reserved block count
#define TFS4_XSR_LLD_TOTAL_BLOCK            1024    //// LLD Total block count

//// BML
#define TFS4_XSR_BML_AVAILABLE_BLOCK_COUNT  ( TFS4_XSR_LLD_TOTAL_BLOCK  \
                                            - TFS4_XSR_LLD_RLRSINRSV \
                                            - TFS4_XSR_BML_PARTITION_INFO)      //// block count

//// BML Partition configuration
#define TFS4_XSR_BML_NUM_OF_PARTITION       2       //// bml partition count

#define TFS4_XSR_BML_PARTITION0_ID          PARTITION_ID_FILESYSTEM     //// id for partition 0
#define TFS4_XSR_BML_PARTITION0_ATTR        BML_PI_ATTR_RW              //// attribute for partitino 0
#define TFS4_XSR_BML_PARTITION0_1STVBN      0                           //// first block number
#define TFS4_XSR_BML_PARTITION0_NUMOFBLOCKS 800                         //// block count

#define TFS4_XSR_BML_PARTITION1_ID          PARTITION_ID_FILESYSTEM1    //// id for partition 1
#define TFS4_XSR_BML_PARTITION1_ATTR        BML_PI_ATTR_RW              //// attribute for partition 1
#define TFS4_XSR_BML_PARTITION1_1STVBN      800                         //// first block number
#define TFS4_XSR_BML_PARTITION1_NUMOFBLOCKS (TFS4_XSR_BML_AVAILABLE_BLOCK_COUNT \
                                            - TFS4_XSR_BML_PARTITION1_1STVBN)
```

Figure 4-28. tfs4_pdev_nand_xsr.h – common NAND and XSR configuration



```
#elif defined (TFS4_XSR_NAND_KFG1G16)
    //// OneNAND, 1Gbit
    /// STL
    #define TFS4_XSR_STL_BLOCKS_PER_UNIT        1
    #define TFS4_XSR_STL_NUM_OF_RSVD_UNIT       10      //// reserved unit count
    #define TFS4_XSR_STL_ASYNC_MODE             FALSE32 //// TRUE32:ASYNC MODE, FALSE32:SYNC MODE
    #define TFS4_XSR_STL_SAM_BUFFER_FACTOR      100     //// Sam buffer factor

    //// ILD
    #define TFS4_XSR_ILD_RSVBLKS                20      //// ILD Reserved block count
    #define TFS4_XSR_ILD_TOTAL_BLOCK            1024    //// ILD Total block count

    //// BML
    #define TFS4_XSR_BML_PARTITION_INFO         3       //// Blocks for partition information
    #define TFS4_XSR_BML_AVAILABLE_BLOCK_COUNT  ( TFS4_XSR_ILD_TOTAL_BLOCK  \
                                                - TFS4_XSR_ILD_RSVBLKS \
                                                - TFS4_XSR_BML_PARTITION_INFO)      //// block count

    //// BML Partition configuration
    #define TFS4_XSR_BML_NUM_OF_PARTITION       2       //// bml partition count

    #define TFS4_XSR_BML_PARTITION0_ID          PARTITION_ID_FILESYSTEM     //// id for partition 0
    #define TFS4_XSR_BML_PARTITION0_ATTR        BML_PI_ATTR_RW              //// attribute for partitino 0
    #define TFS4_XSR_BML_PARTITION0_1STVBN      0                           //// first block number
    #define TFS4_XSR_BML_PARTITION0_NUMOFBLOCKS 800                         //// block count

    #define TFS4_XSR_BML_PARTITION1_ID          PARTITION_ID_FILESYSTEM1    //// id for partition 1
    #define TFS4_XSR_BML_PARTITION1_ATTR        BML_PI_ATTR_RW              //// attribute for partition 1
    #define TFS4_XSR_BML_PARTITION1_1STVBN      800                         //// first block number
    #define TFS4_XSR_BML_PARTITION1_NUMOFBLOCKS (TFS4_XSR_BML_AVAILABLE_BLOCK_COUNT \
                                                - TFS4_XSR_BML_PARTITION1_1STVBN)
```

Figure 4-29 STL and BML configuration

For more information of XSR configuration, refer to a XSR porting guide or XSR programmer's guide. In this chapter, some XSR configuration will be covered. However they are strictly limited to what is related to TFS4.

- TFS4_NAND_SECTOR_SIZE

It is the sector size of NAND Device. It is normally 512 bytes.

- TFS4_NAND_SECTOR_SIZE_BITS

It is the least number of bits to represent `TFS4_NAND_SECTOR_SIZE`.

- TFS4_NAND_START_SECTOR

It is a starting sector address of the NAND region TFS4 uses. It is the logical information. The number of sector filesystem uses is retrieved by `STL_Open()`. It can be set as 0, if you want to use the whole NAND.

- TFS4_NAND_PAGE_SIZE

It is the page size of the NAND.

- TFS4_XSR_STL_BLOCKS_PER_UNIT

It is the number of blocks per unit of `STLConfig` structure. `STLConfig` is a parameter of `STL_Format()`. This is the detailed description of `nBlksPerUnit`.

Unit is an abstract concept to block. Normally, unit is composed of N blocks and the smallest erasable unit. The number of block per unit is configured by `nBlksPerUnit`. For large block NAND device, `nBlksPerUnit` should be only 1, and for small block NAND device, `nBlksPerUnit` can be specified among 1, 2, or 3. As the unit value is increased, the more memory should be reserved for memory, but write performance of random access is more improved.

- TFS4_XSR_NUM_OF_RSVD_UNIT

It is the number of reserved units of `STLConfig` structure. `STLConfig` is a parameter of `STL_Format()`. This is the detailed description of `nNumOf RsvUnits`.

STL manages the whole memory space as a unit. Some of the all units are unusable, which is called a reserved unit. It is configurable and should be more than 2 at least. As the reserved unit value is increased, usable disk capacity for user is decreased, but the write operation is improved.

- TFS4_XSR_STL_ASYNC_MOD

It is used for the parameter of `STL_open()`. `It specify STL running mode. TFS4 does not support ASYNC MODE. It should be FALSE32.`

- TFS4_XSR_LLD_BLKSINRSV

It is the number of reserved blocks for NAND device

- TFS4_XSR_LLD_TOTAL_BLOCK

It is the number of blocks.

- TFS4_XSR_BML_PARTITION_INFO

It is the number of blocks for BML partition info. XSR v1.4.0 uses three blocks for partition information.

- TFS4_XSR_BML_NUL_OF_PARTITION

It is the number of partition in NAND device. It is used for formatting BML. It should be configured by used.

- TFS4_XSR_BML_PARTTIONx_ID, TFS4_XSR_BML_PARTITIONx_ATTR, TFS4_XSR_BML_PARTITIONx_1STVBN, TFS4_XSR_BML_PARTITIONx_NUMOFBLOCKS

They are the parameters of `XSRPartI` structure; `XSRPartI` is used as the parameter of `BML_Format()`.

`BML_Format()` is called once each target. It doesn't need to be used.
Use `BML_REPARTITION` to modify the BML partition information, instead of `BML_INIT_FORMAT`.

For details about BML, refer to BML programmer's guide.

4.2.5.7. TFS4 Interface with MMC(or HSMMC) Host Device Driver

TFS4 needs a MMC(or HSMMC) Host Device Driver to use a MMC(or HSMMC). TFS4 includes a sample MMC(or HSMMC) host device driver, which is implemented based on ReindeerPlus.

But you have to modify it according to your target clock or bus setting.

For physical device interface of TFS4, the following interfaces are necessary.

Table 18. Necessary Physical Device Interface for TFS4

Function	Description
Init	It initializes a device.
Read Status	It retrieves a device status.
Open	It opens a device.
Close	It closes a device.
Read	It reads data from device.
Write	It writes data on device.
Erase	It erases specific region of device.
IO control	It performs a generic IO control.

If you try to write a MMC(or HSMMC) host device driver on your target, you should implement the functions that provides the features mentioned above.

< External Card Insert/Remove Notification >
Interrupt occurs when the external device is being inserted or ejected.
If an external device is inserted, external device should be registered to use. To register a device to TFS4 use tfs4_pdev_reg(). reference to TFS4 programmers guide for detailed information.
If an external device is disconnected. External device should be un-registered from TFS4. To un-register an external device use tfs4_pdev_unreg().

All of the above features can be implemented in tfs4_pdev_mmc_nucleus.c/h; the file name can be changed by user.

Table 19. Implemented Functions for MMC(or HSMMC) Host Device Driver

Function	Description
`t_int32 tfs4_pdev_mmc_init_device(void)`	It initializes a device. It retrieves the MMC(or HSMMC) information by using `mmc_get_stat()` and stores it in `tfs4_mmc_info`. For details about that, refer to 4.2.2. MMC(or HSMMC) Host Device Driver Development.

	It specifies a function pointer to use the MMC(or HSMMC) device.
`static t_int32 _tfs4_mmc_read_status(t_physical_device_info *pDI)`	It retrieves a device status by storing into `t_physical_device_info`. This function be used while tfs4_pdev_reg()
`static t_int32 _tfs4_mmc_open(void)`	It opens a device.
`static t_int32 _tfs4_mmc_close(void)`	It closes a device.
`static t_int32 _tfs4_mmc_read_sector(t_uint32 dwSectorNo, t_uint8 *pBuff, t_uint32 dwCount)`	It reads data from device. It reads the `dwCount` number of sector from `dwSectorNo` and writes a code to store the sector to `pBuff`. It returns the number of read sector on success and `TFS4_EIO` on failure.
`static t_int32 _tfs4_mmc_write_sector(t_uint32 dwSectorNo, t_uint8 *pBuff, t_uint32 dwCount)`	It writes data on device. It writes the `dwCount` number of sectors into flash memory space starting at `dwSectorNo`. It returns the number of written sectors on success and `TFS4_EIO` on failure.
`static t_int32 _tfs4_mmc_erase_sector(t_uint32 dwSectorNo, t_uint32 dwCount)`	It erases the `dwCount` number of sectors from `dwSectorNo` in the flash memory. It returns 0 on success and `TFS4_EIO` on failure.
`static t_int32 _tfs4_mmc_ioctl(t_int32 command, void *pBuff)`	It performs a generic IO control. You can do additional features on it. Currently, only the function of retrieving MMC(or HSMMC) information is implemented.

Note

Read/write function of external device driver should include a routine to check byte alignment, because TFS4 just uses the address received from an application.

4.3. Build with Target OS

If you've performed TFS4 porting as explained in the previous chapter successfully, now you can build them together.

To build TFS4 with OS, you first need to compose a project file, makefile, or something to build according to your build tool so that all the components are built together. Your project file has to include:

- TFS4 library and configured TFS4 sources
- XSR library and sources
- MMC(or HSMMC) library or sources (Optional)
- OS library (it is Nucleus in this document)

The following table lists the target-dependant source files of TFS4.

Table 20. Configurable File List of TFS4

File Name	Description
tfs4_errno.c	It stores a TFS4 error number or returns the stored error number.
tfs4_memory.c	It has the TFS4 memory related functions.
tfs4_pdev_nand_xsr.c	It has the code implemented to access to NAND device through XSR.
tfs4_pdev_mmc_nucleus.c	It interfaces a MMC(or HSMMC) host device driver running on Nucleus.
tfs4_semaphore.c	It has the codes implemented for semaphore.
tfs4_time.c	It has the codes implemented for retrieving the time information that filesystem uses.
tfs4_tty.c	It is used for TFS4 test shell. It performs the I/O control through UART or keyboard. It is used only during development.

The following table lists the XSR source files that are ported to ReindeerPlus. For other targets, the file names can be changed.

Table 21. Configurable File List of XSR

File Name	Description

PAM.cpp	Platform Adaptation Module
PNL.cpp	Low Level Device Driver for large block NAND
NucleusOAM.cpp	It has the code implemented for XSR porting to Nucleus.

The following picture shows the implemented MMC(or HSMMC) host device driver files.

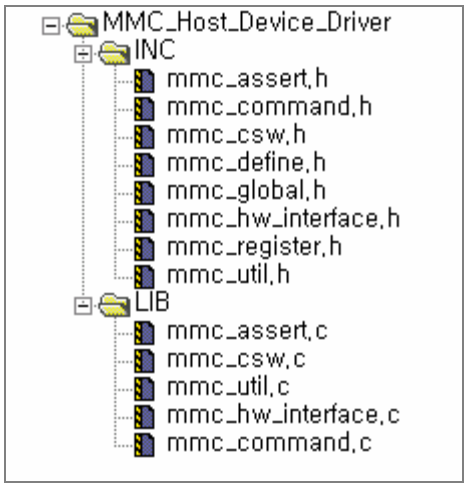


Figure 4-30. MMC(or HSMMC) Host Device Driver File List

The following table represents TFS4 library and source to build with OS.

Table 22. TFS4 Library & Sources

	TFS4	**XSR**	**MMC(or HSMMC) Host Driver**
Library	tfs4_lib.a	XSR32lv4.lib	
Source	tfs4_errno.c tfs4_memory.c tfs4_pdev_mmc_reindeep_plus.c tfs4_pdev_nand_xsr.c tfs4_semaphore.c tfs4_time.c tfs4_tty.c tfs4_tuning.c	NucleusOAM.cpp PAM.cpp PNL.cpp	mmc_assert.c mmc_command.c mmc_csw.c mmc_hw_interface.c mmc_util.c

tfs4_integration_test.c/h includes a test shell for TFS4 test. If you don't need the test shell, do not add the tfs4_integration_test.c/h to the project file.

The following pictures show Reindeer_Plus_With_TFS4.mcp to build TFS4 with XSR, MMC(or HSMMC) host device driver library, and OS.

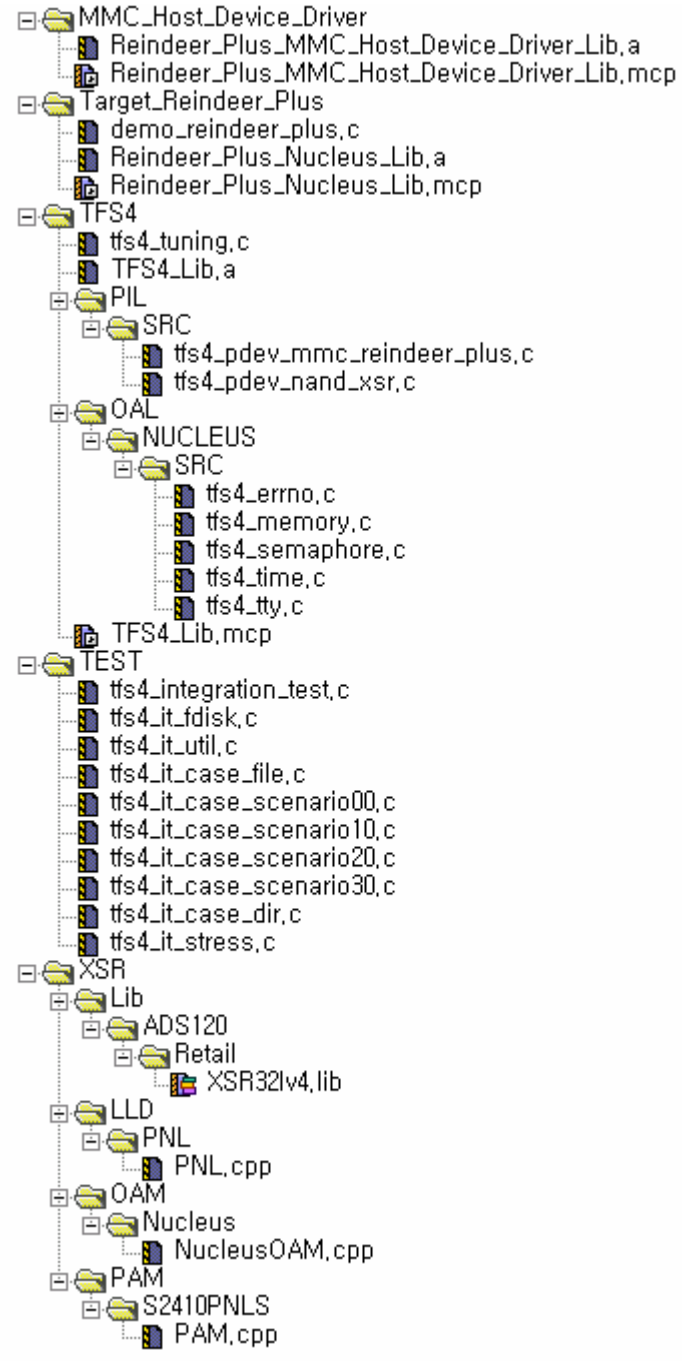


Figure 4-31. Reindeer_Plus_With_TFS4.mcp

< TFS4 build with Nucleus by using the project file on ADS v1.2 >

The following picture shows the TFS4 directory structure.

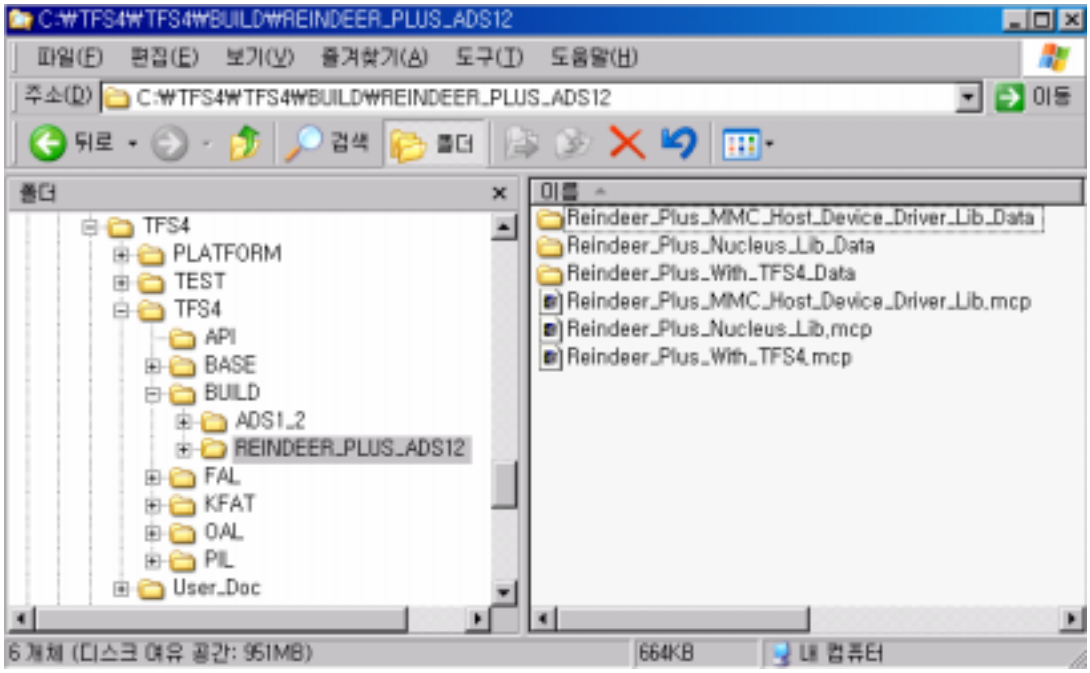


Figure 4-32. Directory Path of TFS4-related Project Files

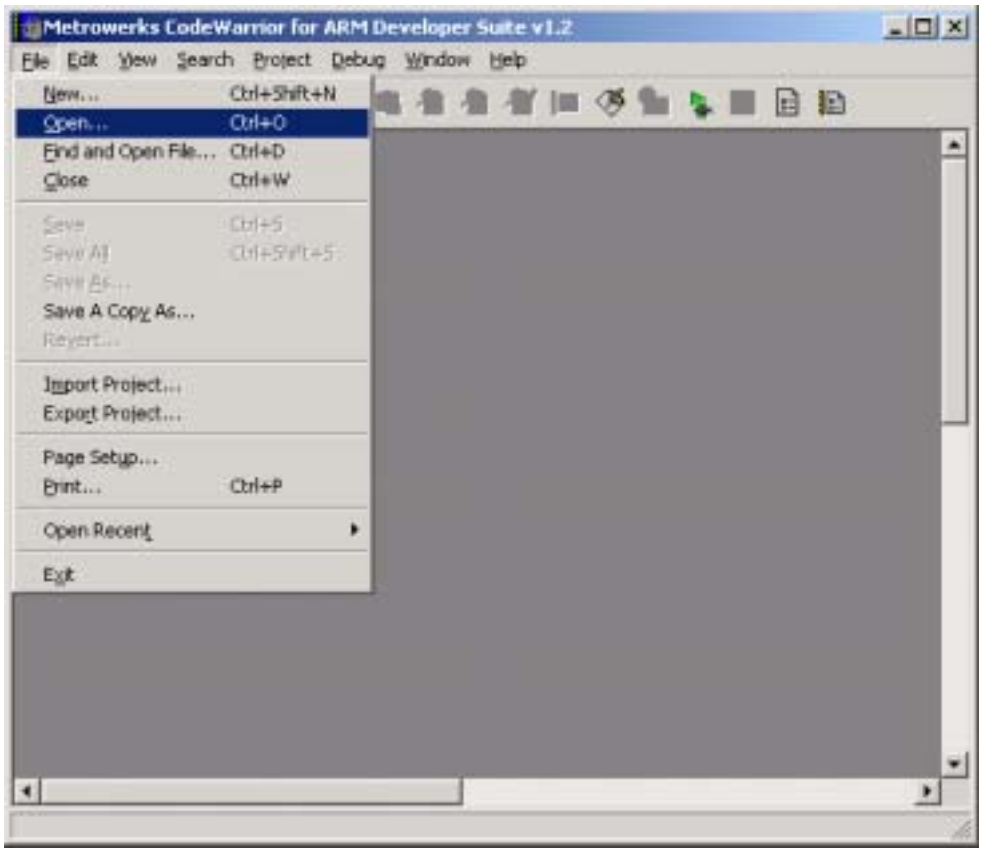
If you use the ADS1.2 build tool and the Nucleus OS, you can use the provided project file, “Reindeer_Plus_With_TFS4.mcp” and “Reindeer_Plus_With_TFS4_With_HS_MMC.mcp” in the ”C:\TFS4\TFS4\BUILD\REINDEER_PLUS_ADS12” directory, as shown in the above picture.

Currently, the “Reindeer_Plus_With_TFS4.mcp” and “Reindeer_Plus_With_TFS4_With_HS_MMC.mcp” project file includes:

- TFS4 library and configured TFS4 sources
- XSR library and sources
- MMC (or HSMMC) library and sources
- OS library (it is Nucleus in this document)

Here, we explain the TFS4 build with OS by using ADS1.2.

1. Open the project file for making a TFS4 library. You can click “File” → “Open” on the menu bar of the screen as follows.

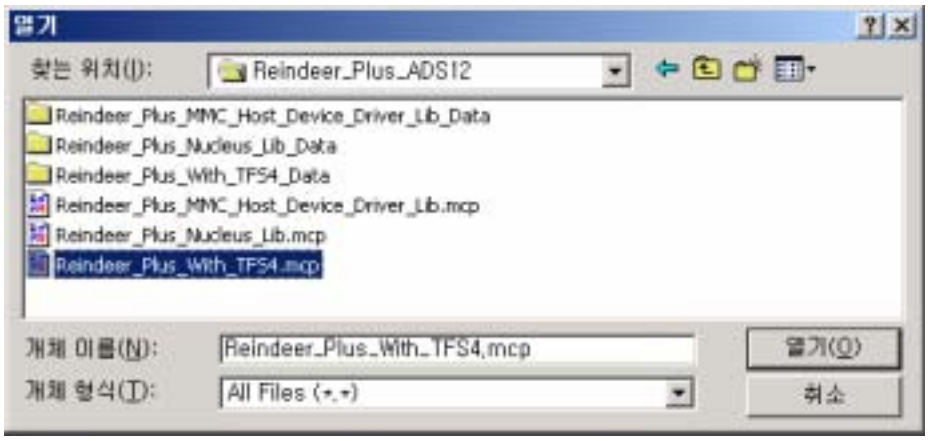


Or,  button on icon bar.

Note

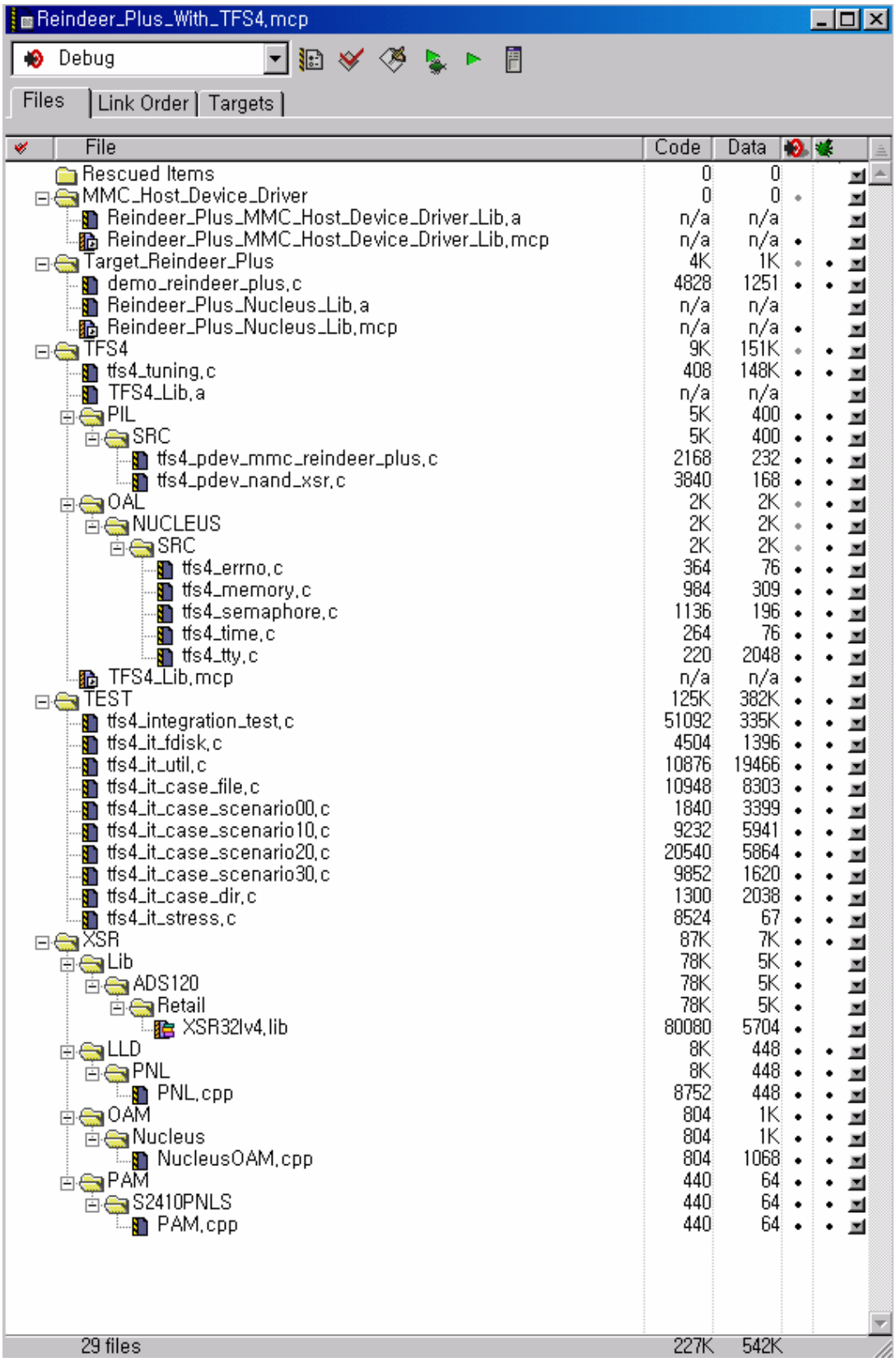
We provide the project file for building TFS4 with XSR, sample MMC (or HSMMC) host device driver, and adaptation layer for Nucleus. The name of the project file is Reindeer_Plus_With_TFS4.mcp (or Reindeer_Plus_With_TFS4_With_HS_MMC.mcp); the “mcp” is the extension of ADS. If you don’t use the ADS v1.2, you need to create your own project file.

2. The screen to open a file appears.



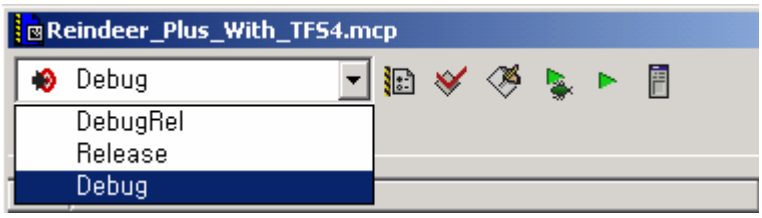
Find the Reindeer_Plus_With_TFS4.mcp file and press “Open” button on the screen.

3. The Reindeer_Plus_With_TFS4.mcp file is opened as below.


Reindeer_Plus_With_TFS4.mcp
Debug
Files
Link Order
Targets
File
Code
Data
Rescued Items 0 0
MMC_Host_Device_Driver 0 0
Reindeer_Plus_MMC_Host_Device_Driver_Lib.a n/a n/a
Reindeer_Plus_MMC_Host_Device_Driver_Lib.mcp n/a n/a
Target_Reindeer_Plus 4K 1K
demo_reindeer_plus.c 4828 1251
Reindeer_Plus_Nucleus_Lib.a n/a n/a
Reindeer_Plus_Nucleus_Lib.mcp n/a n/a
TFS4 9K 151K
tfs4_tuning.c 408 148K
TFS4_Lib.a n/a n/a
PIL 5K 400
SRC 5K 400
tfs4_pdev_mmc_reindeer_plus.c 2168 232
tfs4_pdev_nand_xsr.c 3840 168
OAL 2K 2K
NUCLEUS 2K 2K
SRC 2K 2K
tfs4_errno.c 364 76
tfs4_memory.c 984 309
tfs4_semaphore.c 1136 196
tfs4_time.c 264 76
tfs4_tty.c 220 2048
TFS4_Lib.mcp n/a n/a
TEST 125K 382K
tfs4_integration_test.c 51092 335K
tfs4_it_fdisk.c 4504 1396
tfs4_it_util.c 10876 19466
tfs4_it_case_file.c 10948 8303
tfs4_it_case_scenario00.c 1840 3399
tfs4_it_case_scenario10.c 9232 5941
tfs4_it_case_scenario20.c 20540 5864
tfs4_it_case_scenario30.c 9852 1620
tfs4_it_case_dir.c 1300 2038
tfs4_it_stress.c 8524 67
XSR 87K 7K
Lib 78K 5K
ADS120 78K 5K
Retail 78K 5K
XSR32lv4.lib 80080 5704
LLD 8K 448
PNL 8K 448
PNL.cpp 8752 448
OAM 804 1K
Nucleus 804 1K
NucleusOAM.cpp 804 1068
PAM 440 64
S2410PNLS 440 64
PAM.cpp 440 64
29 files 227K 542K

See the source and library files of Nucleus OS ported on ReindeerPlus, XSR, MMC (or HSMMC) Host Device Driver, and TFS4 to build together in the project file.

4. Select a type of build target.



There are three types of build targets. The following describes the meaning of each build target.

- Debug mode: The output binary is compiled with debugging symbols and information of line numbers.
- Release mode: In this configuration, the output binary will be fully optimized and contains no debugging symbols.
- DebugRel mode: Adequate optimization level and including minimal debugging information.

5. Press  the build setting button.

6. Specify configurations for your target in the Build Settings dialog window.

But the build setting screen can be a little different depending on the build mode setting you select. You have to consider the language setting your compiler supports in the build setting screen. For ADS v1.2, you need to set the build options for each language; you may not need to do that for other build tool.

The following screens show the necessary build options for each build mode and language; the sample build options are based on ReindeerPlus, ARM CPU, and ADS v1.2. You can set the build options suitable to your target and compile environment by referring to the below sample options.

Note

The build option depends on the compiler.

<Debug settings for ARM Assembler>

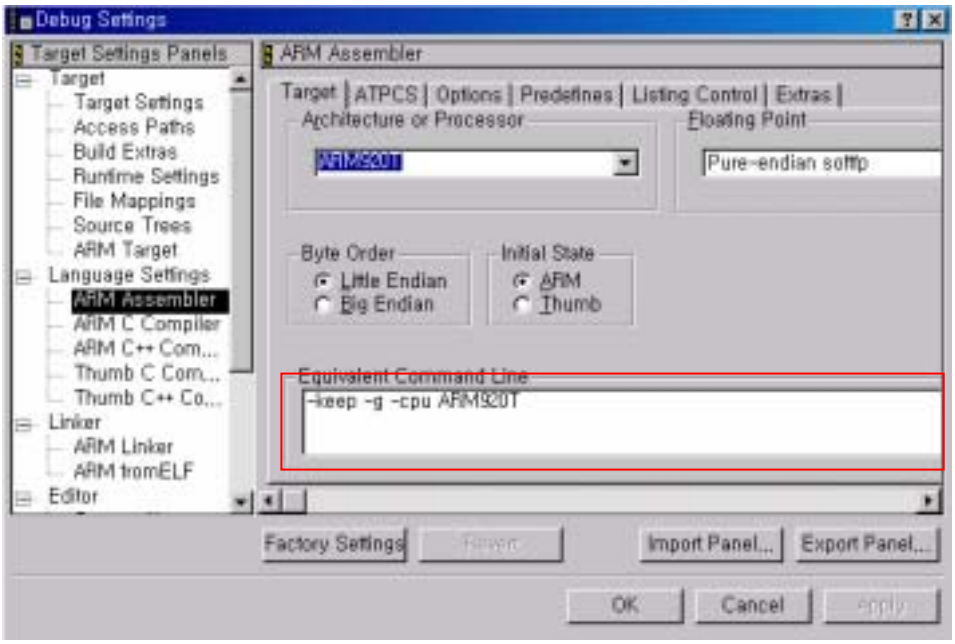


Figure 4-33. Debug Settings for ARM Assembler on ADS v1.2

<Debug settings for ARM C Compiler>

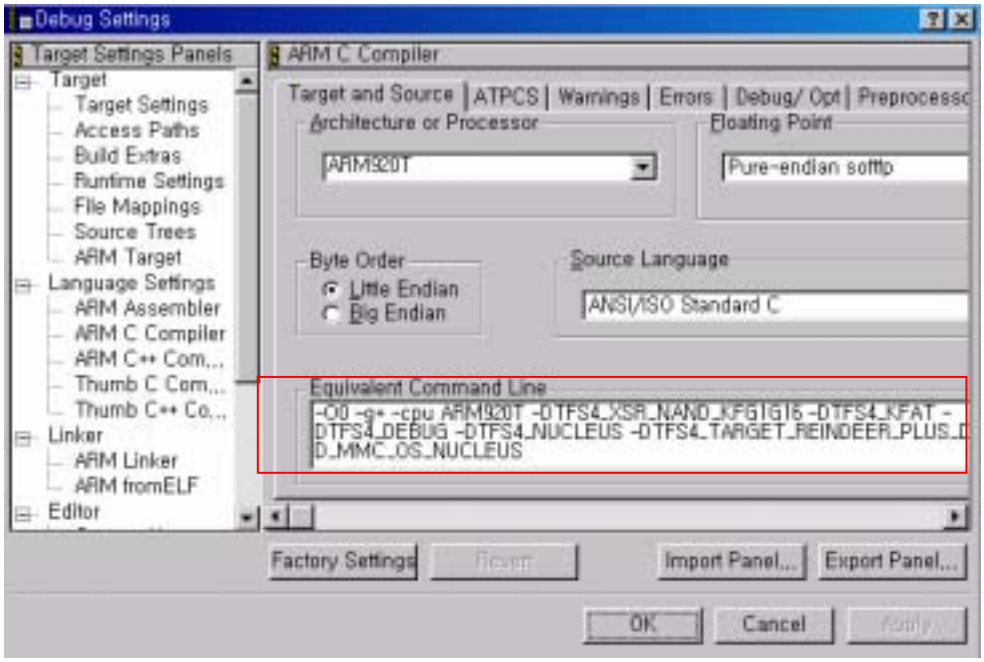


Figure 4-34. Debug Settings for ARM C Compiler on ADS v1.2

<Debug settings for ARM C++ Compiler>

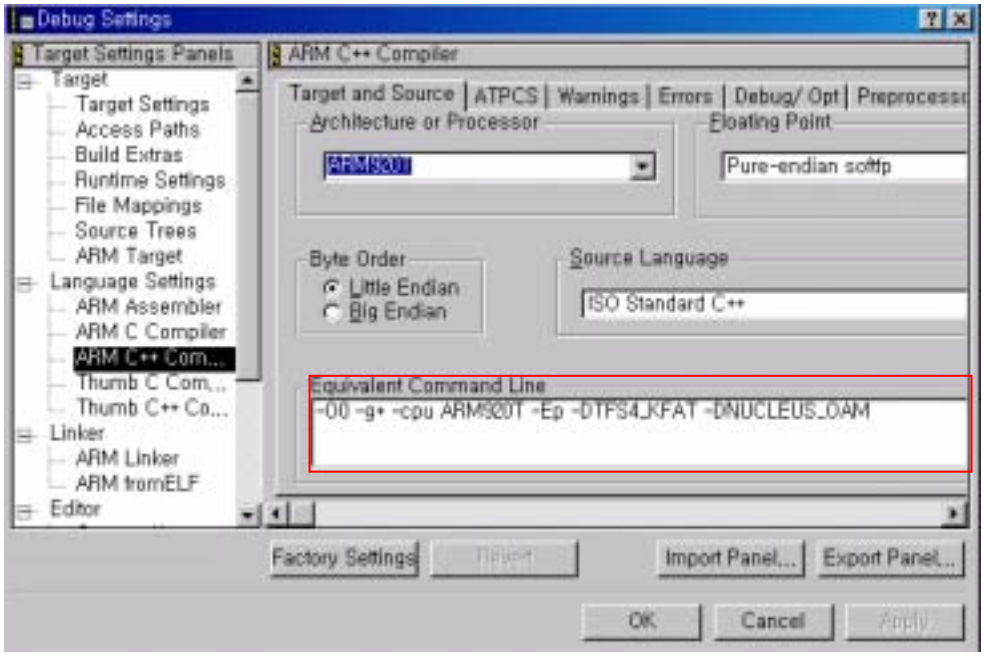


Figure 4-35. Debug Settings for ARM C++ Compiler on ADS v1.2

< Release settings for ARM Assembler>

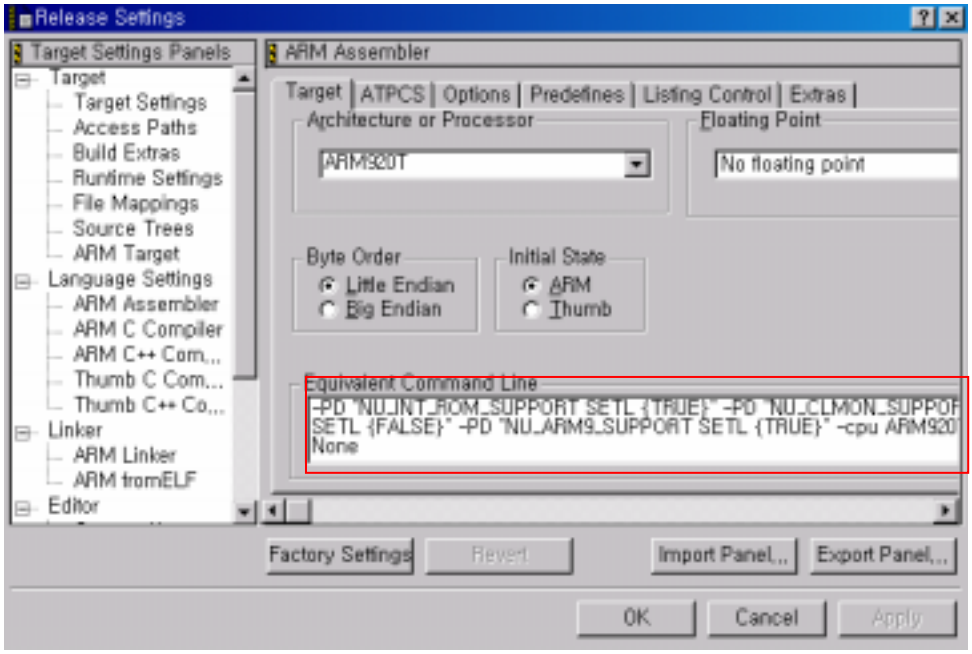


Figure 4-36. Release Settings for ARM Assembler on ADS v1.2

< Release settings for C Compiler>

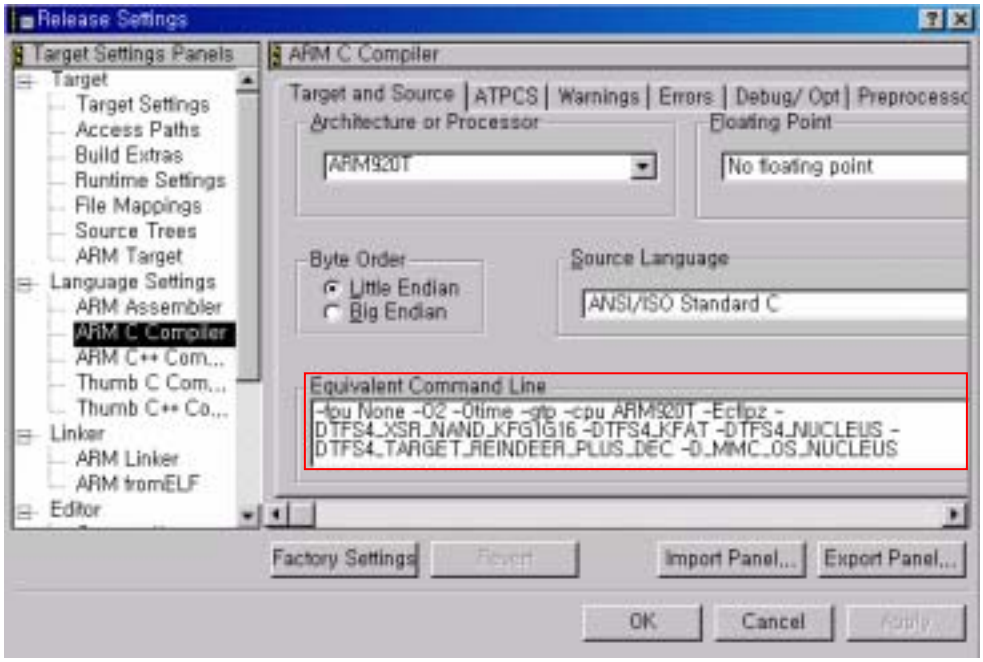


Figure 4-37. Release Settings for ARM C Compiler on ADS v1.2

< Release settings for C++ Compiler>

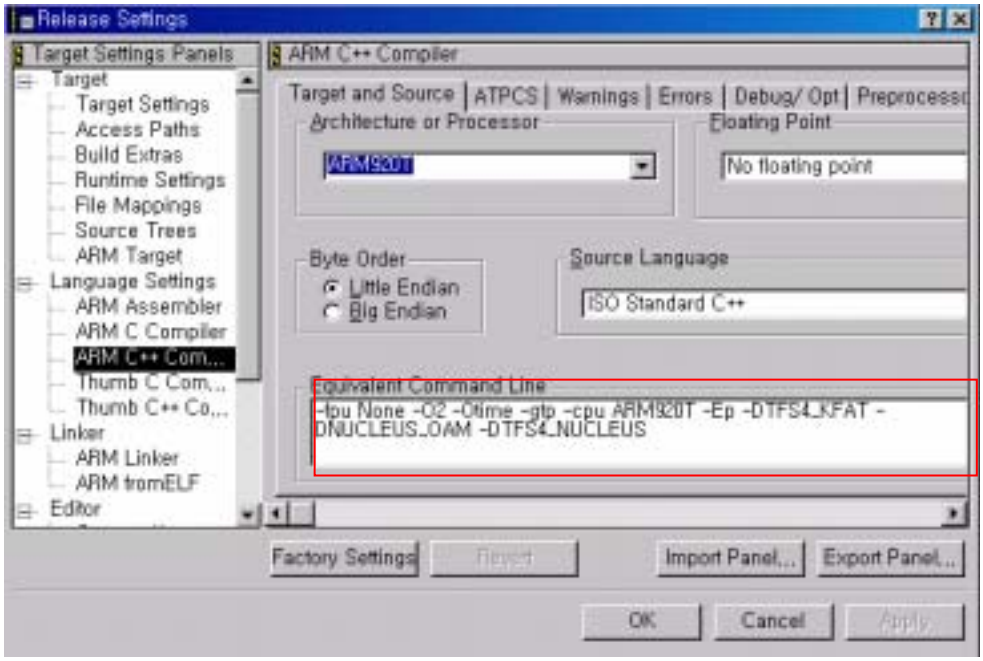


Figure 4-38. Release Settings for ARM C++ Compiler on ADS v1.2

You have to add build options for your target and compile environment.

7. Additionally, you need to add the access paths on the build settings to include the TFS4-related header files (TFS4\API, TFS4\BASE\INC, TFS4\BASE\UNICODE\INC, TFS4\FAL\INC, TFS4\KFAT\INC, TFS4\OAL\NUCLEUS\INC, TFS4\PIL\INC, XSR header file path, and MMC(or HSMMC) host device driver header file path, etc) while TFS4 is compiled.

If you define the header file at tfs4_config.h, you don't have to add the include path to the build setting.

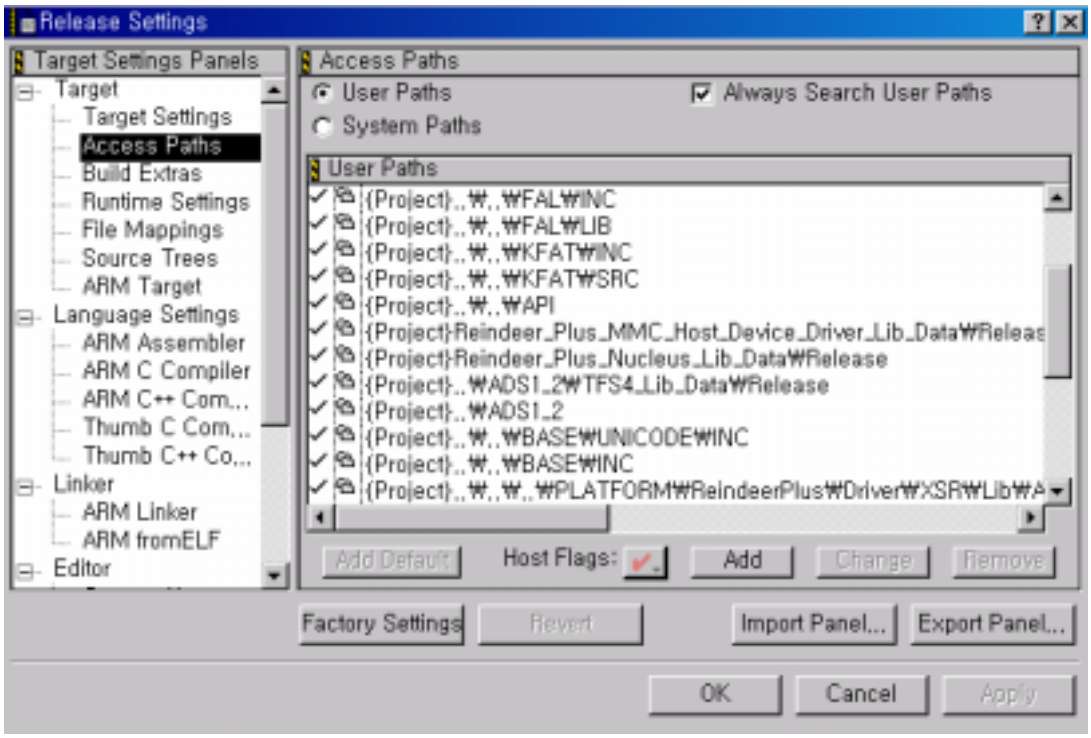
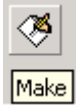


Figure 4-39. Include Access Paths

8. After setting the build options, press "OK" button to save.

9. Press 

button on ADS v1.2.

10. The project file build is performed; the project file includes TFS4 and OS. The target image is created on your host.

You finished the build of the ported TFS4 and OS together. Now you can transfer the target image to your target.

4.4. Download to Target Device

We assume your host and target is already connected to each other by using Multi-ICE (other ICE device can be substituted for it). Or you can write the target image on NAND flash memory. You can select the deployment type of the target image, depending on your development progress.

This is step for downloading the target image to SDRAM.

1. Power on your target.

2. Execute the AXD debugger of ADS 1.2 on your host.

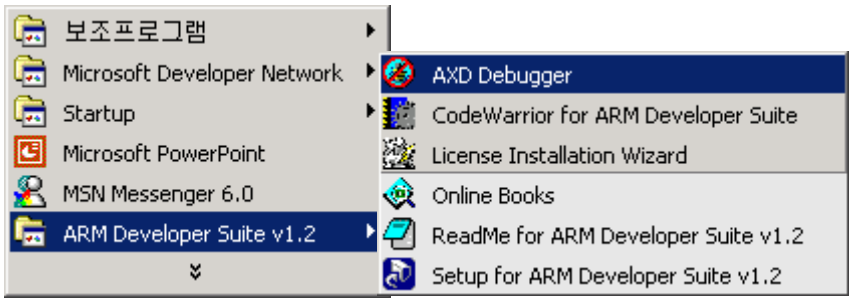


Figure 4-40. Execute AXD Debugger of ADS v1.2

3. Execute the code for target initialization as follows.

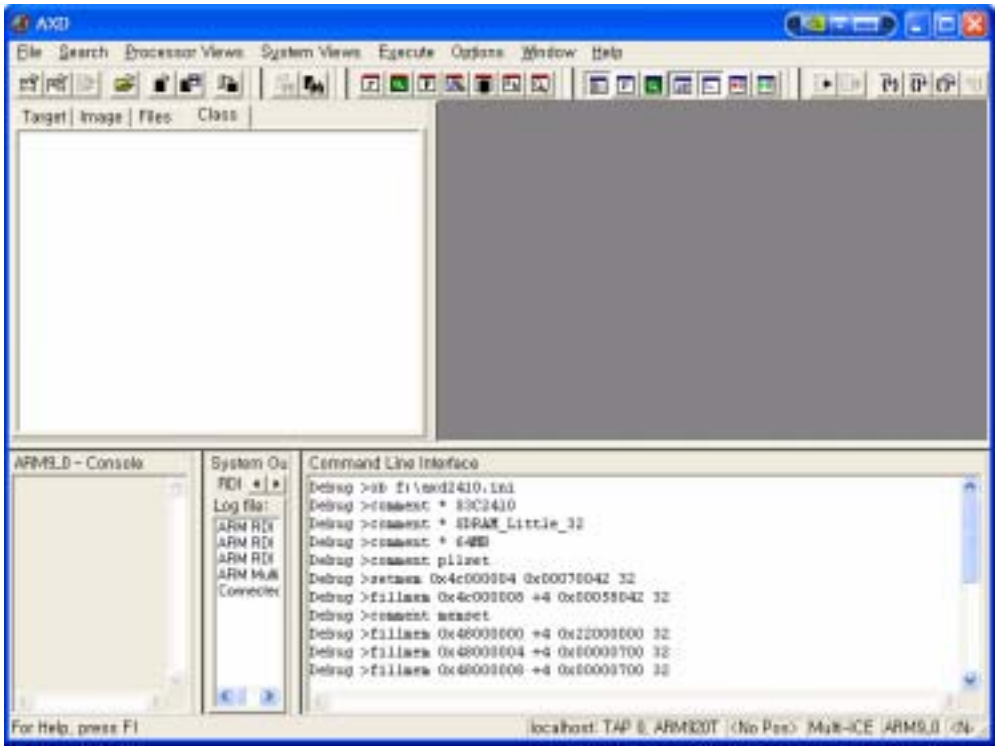


Figure 4-41. Initialize a target

4. Press “File” → “Load Image” on menu to select the target image to download.

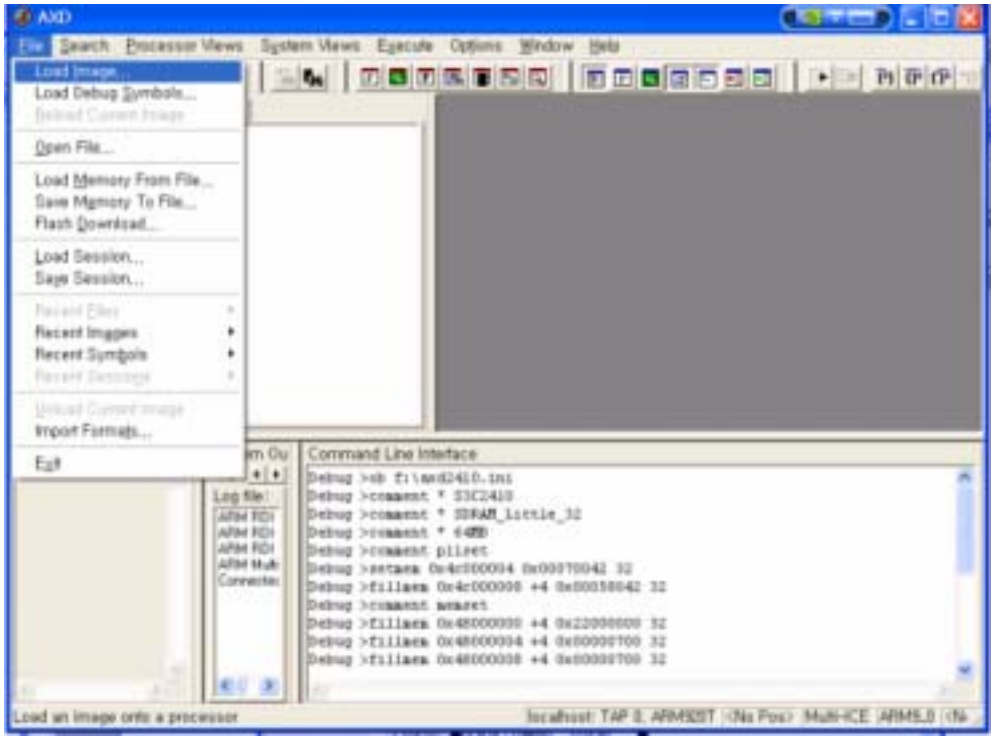


Figure 4-42. Press Load Image Button

5. The following screen appears.

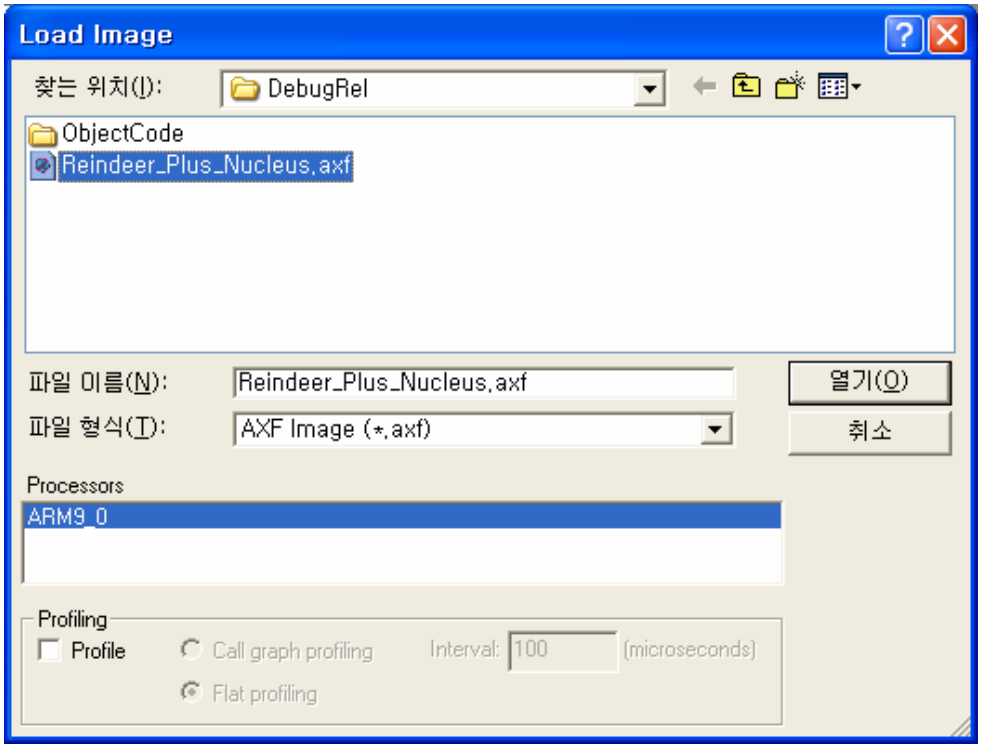


Figure 4-43. Search the Image Being Loaded

Select the target image file to download.

6. The screen shows the target image is getting loaded to target.

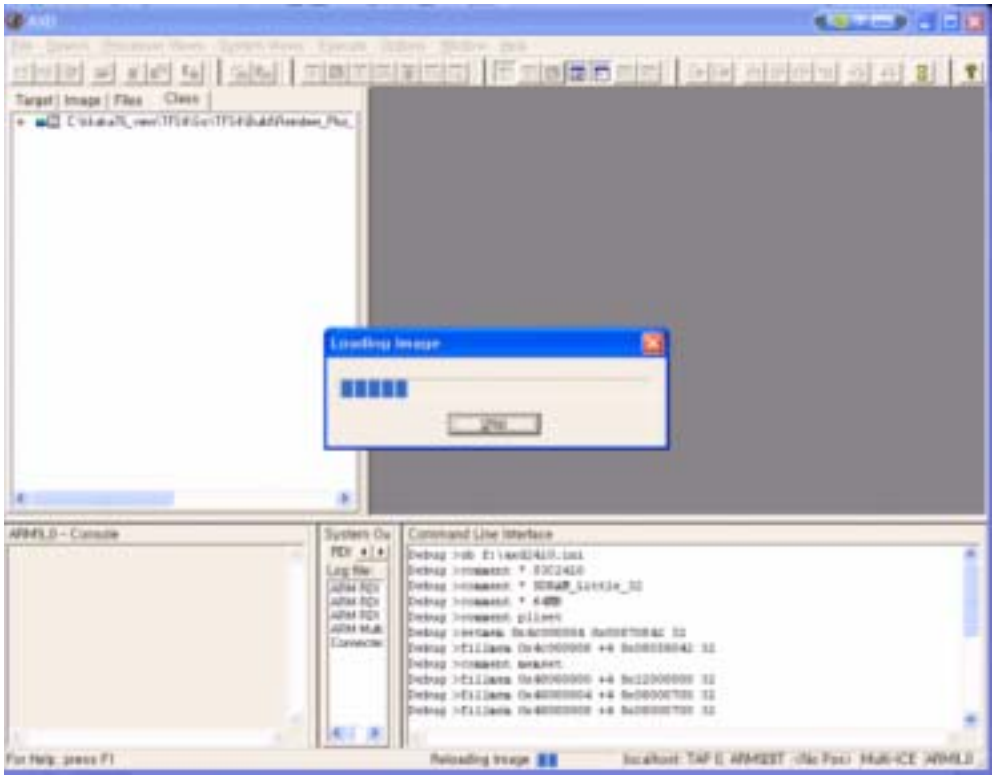


Figure 4-44. Load the Image

7. The screen shows that the downloading is finished.

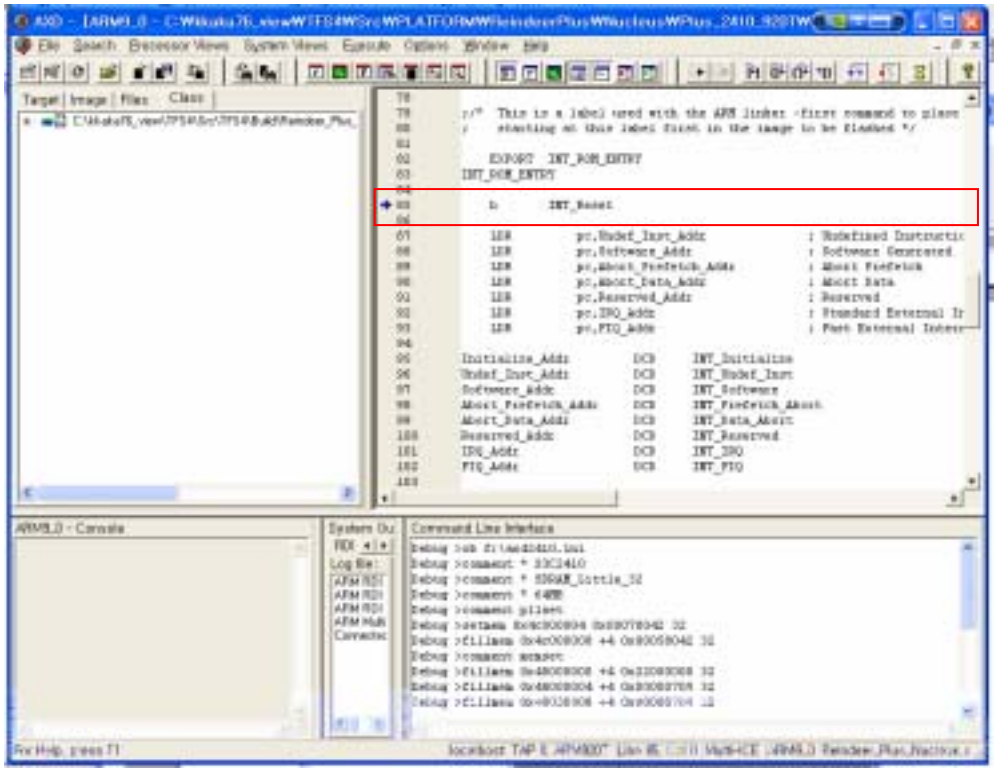


Figure 4-45. Find the Starting Point of the Image

"→" indicates to the starting point of the target image for running on the editor screen.

Now, downloading the image to target is completed and you can execute it on target.

5. TFS4 Test Process

This chapter describes TFS4 test process. If you are developing the TFS4 or other program using TFS4, you can follow this test process.

To test TFS4, you first have to test XSR or MMC (or HSMMC) host device driver, because TFS4 works with them and so you cannot find where the TFS4 error is from if they are not tested before TFS4.

TFS4 test process includes the sub sections as follows.

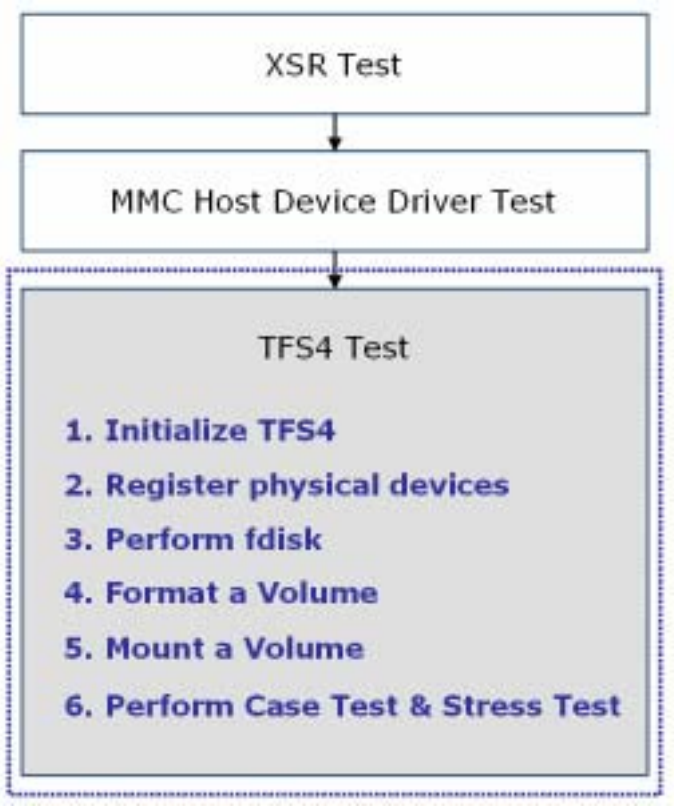


Figure 5-1. TFS4 Test Process

Of the TFS4 Test, the step 1 and 3 don't have to be performed all the time. They can be performed if needed.

You can write a test code or use the provided test shell for XSR, MMC (or HSMMC), and TFS4 test. This chapter explains the TFS4 test by using the test shell.

5.1. XSR Test

For XSR test, you can refer to XSR programmer's guide.
Here, the important thing is that you can test TFS4, only after XSR is guaranteed to work reliably on target.

5.2. MMC (or HSMMC) test

For MMC (or HSMMC) test, you have to perform the sector read/write operation test. If you want to test the HSMMC, before you perform the sector read/write operation test, you have to change the bus width from 1 bit to 4 bit (or 8 bit).

5.3. TFS4 test

TFS4 test is to check basic functionalities of TFS4. It is to find if TFS4 File System creates, reads, or writes a file/directory on NAND flash memory or MMC (or HSMMC).

You can test all the TFS4 APIs listed in TFS4 programmer's guide. The test shell has a lot of commands (they include all the TFS4 APIs), to test the basic features of TFS4.

TFS4 test is performed according to the below sequence.

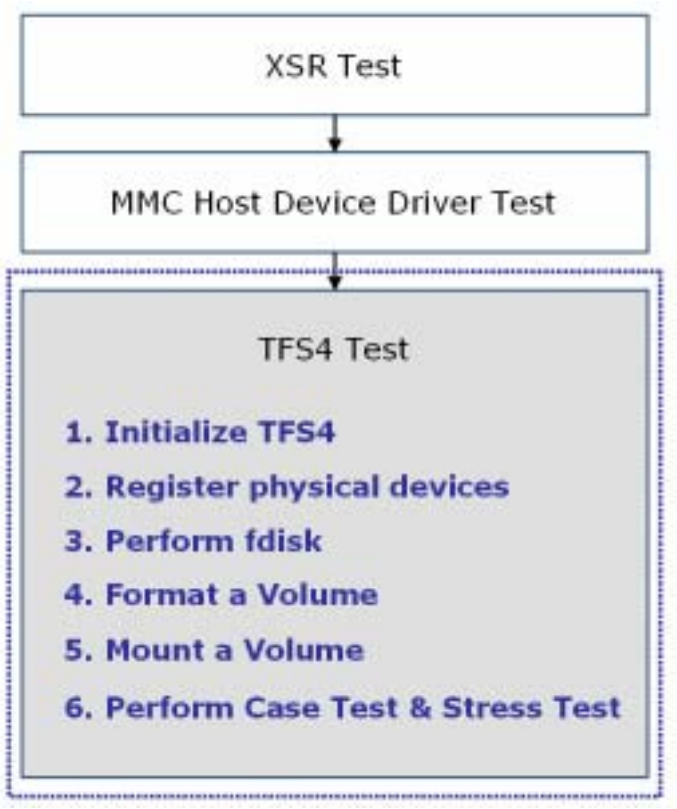


Figure 5-2. TFS4 Test Sequence

To see TFS4 is executed and tested, you should use a terminal program. You can see the TFS4 running state and get the success or failure message from target. Host is connected to target through serial line and shell is running on target for data communication; of course, target device should support UART.

Note

Of the TFS4 Test, the step 3 and 4 don't have to be performed all the time. They can be performed if needed.

< UART settings between host and target >

Before you execute a terminal program on your host, the UART setting should be done for data communication with target.

You can set it before the downloaded target image is executed.

The following picture shows the UART/USB setting screen.

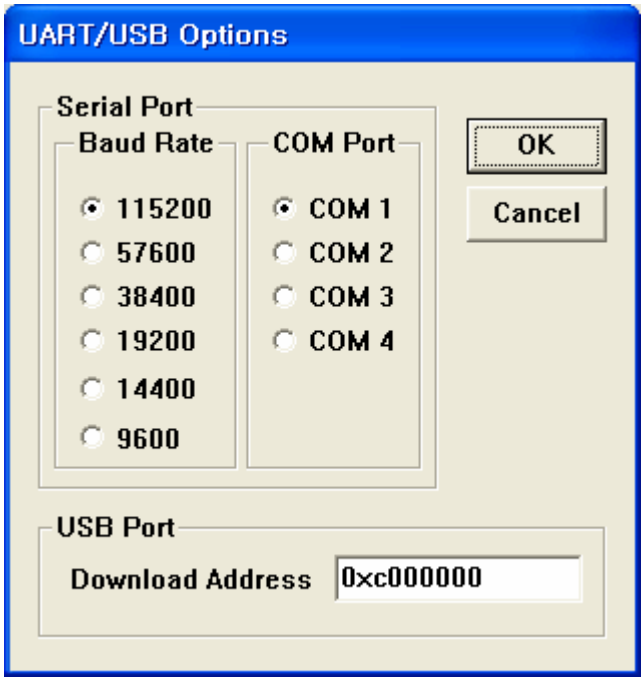


Figure 5-3. UART Options

Check the option suitable to your target UART.

- Baud Rate: it is to set the same transmission speed between host and target. Mostly it is 115200.
- COM Port: PC uses two COM ports, 1 and 2. Normally it is set to 1.

Your terminal prints characters out if you set the UART options successfully. If not, your terminal prints the broken character or nothing.

1. Power on your target.

2. Execute your terminal program on your host.

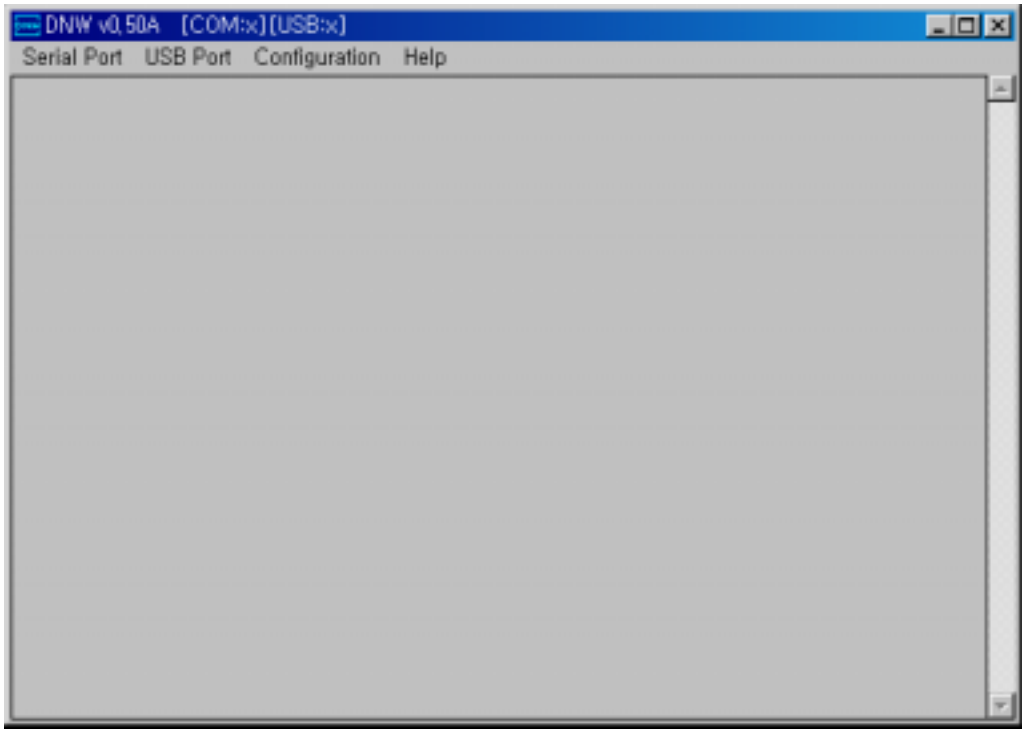


Figure 5-4. Execute a Terminal Program for Test

3. Press “Execute” → “Go” on menu of AXD debugger screen.

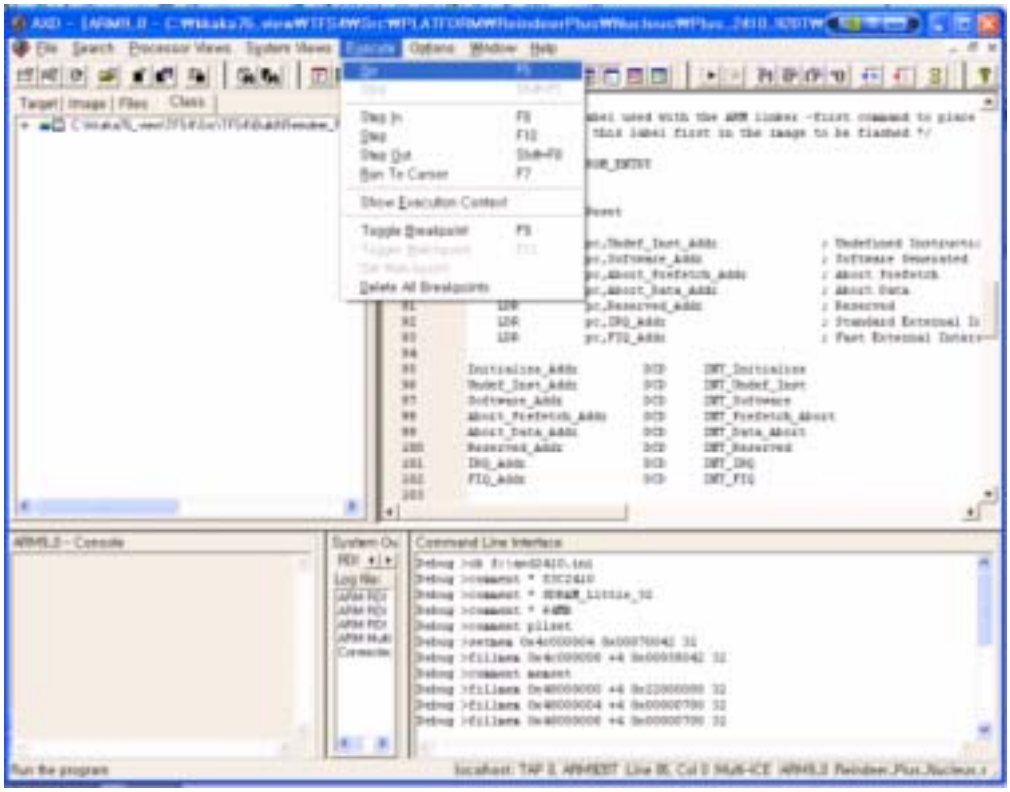


Figure 5-5. Running TFS4

4. The terminal shows the TFS4 test shell is executed.

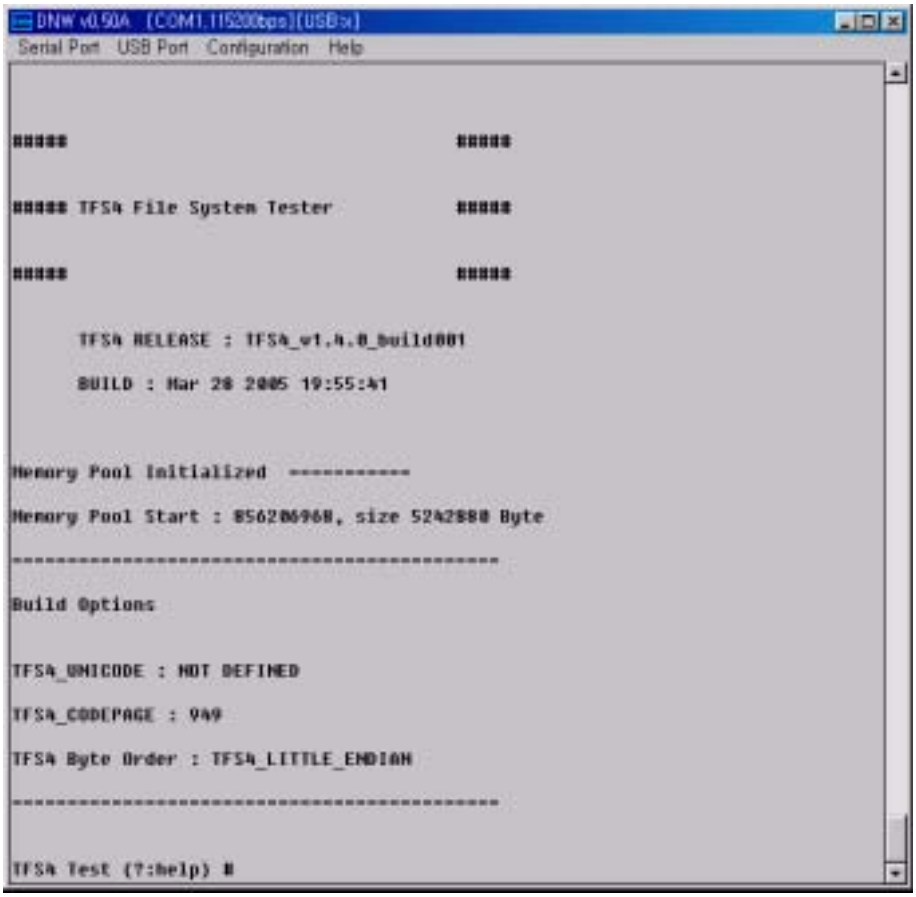


Figure 5-6. TFS4 Test Shell

It waits for user's command. Enter "?" to see the command list.

<BML format information used on ReindeerPlus>

First of all, BML format of XSR should be performed to format a NAND device.
There is the code for BML format in `tfs4_pdev_nand_xsr_bml_format()` of the tfs4_pdev_nand_xsr.c file. You can configure the file if necessary, with reference to XSR documents. BML format should be done once.

XSR partition of the `t_int32 tfs4_pdev_nand_xsr_bml_format()` function should be modified according to target bootloader. In the current release version, we do not consider the bootloader.

The following screen shows performing BML format; BML region is already created in XSR test.

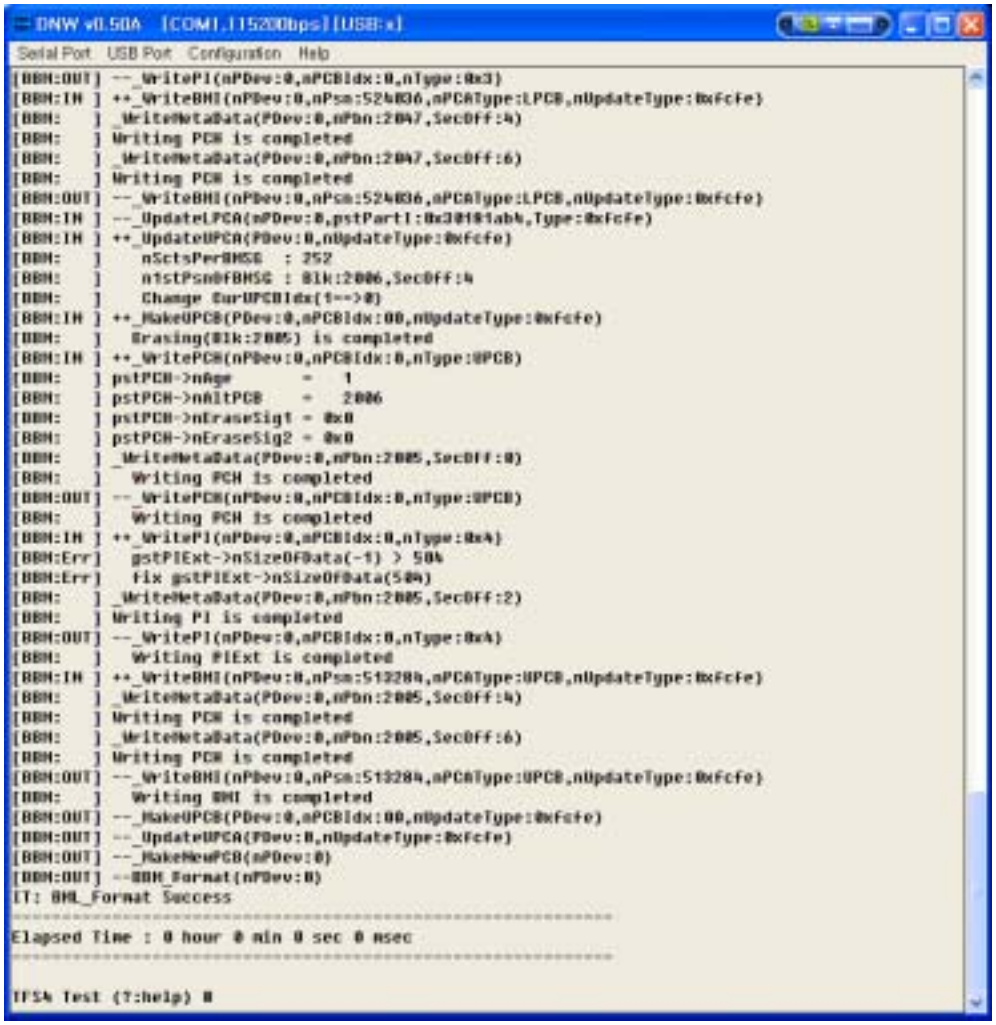


Figure 5-7. Perform BML_format

t_int32 tfs4_pdev_nand_xsr_bml_format(void) is executed in test shell and you can see the message printed on the terminal as follows.

5.3.1. Initialize TFS4

Initializing TFS4 should be performed before using TFS4 after the partition of NAND device or MMC(or HSMMC) is created.

Enter “tfs4_init” on the test shell.

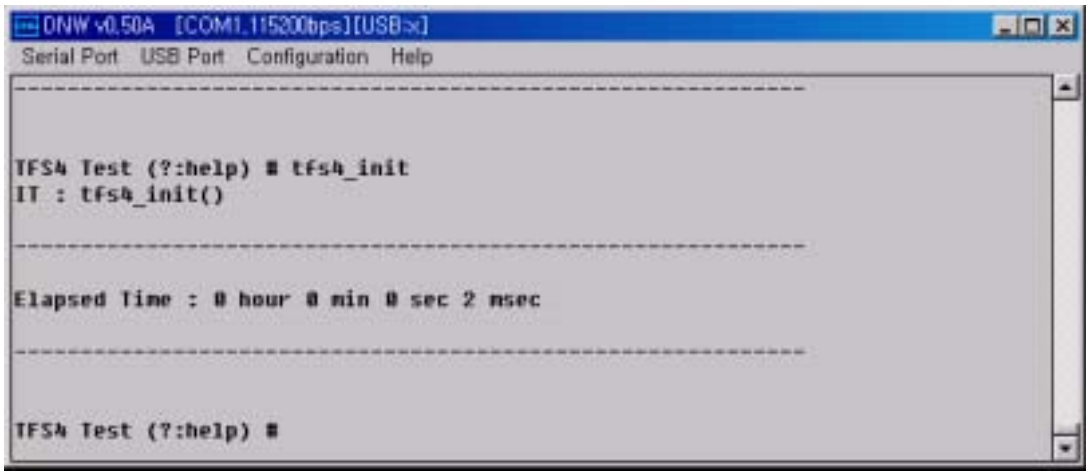
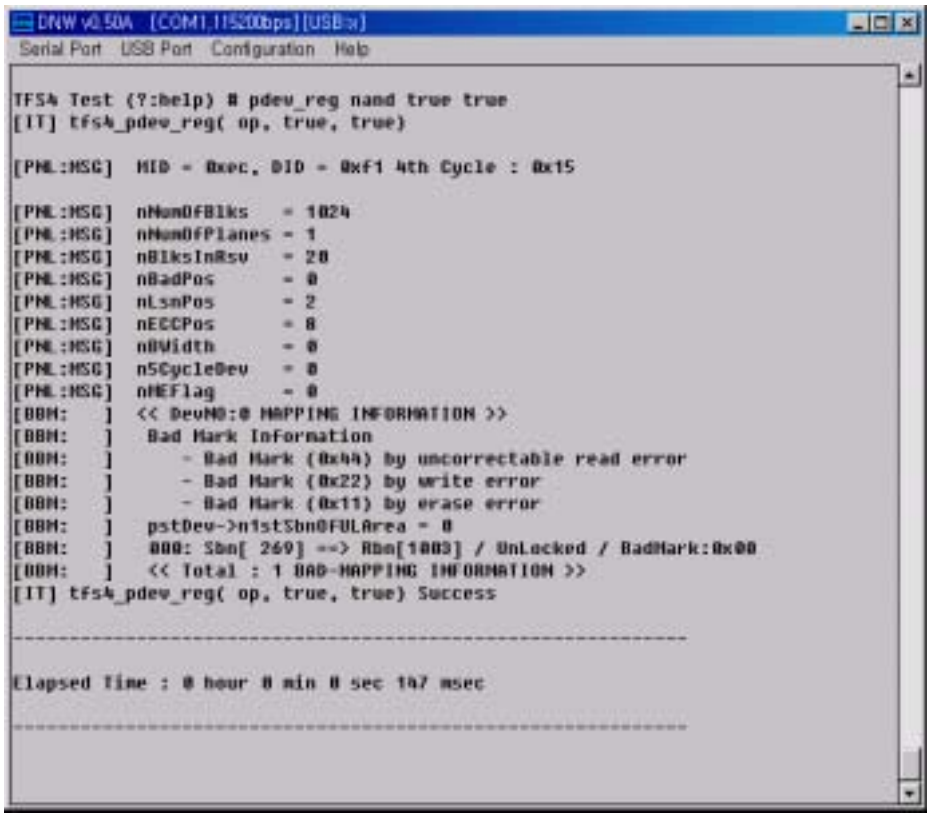


Figure 5-8. Perform tfs4_init

tfs4_init is executed in test shell .

5.3.2. Register a physical device

Register a physical device to TFS4.



5.3.3. Perform fdisk

TFS4 supports making four partitions on NAND device at maximum. You need to set the number of partition for TFS4 to use.

When NAND flash is manufactured, TFS4 is written on NAND flash by using ROM write. Thus, fdisk is used only when development. You can test if the partition is created, deleted, or modified on test shell. TFS4 fdisk is basically same as Linux fdisk.

In case of MMC(or HSMMC) Device, Windows OS supports only one partition. If you create more than one partition, only the first partition is only detected.

1. Enter “tfs4_fdisk {device}” on your host terminal as follows.

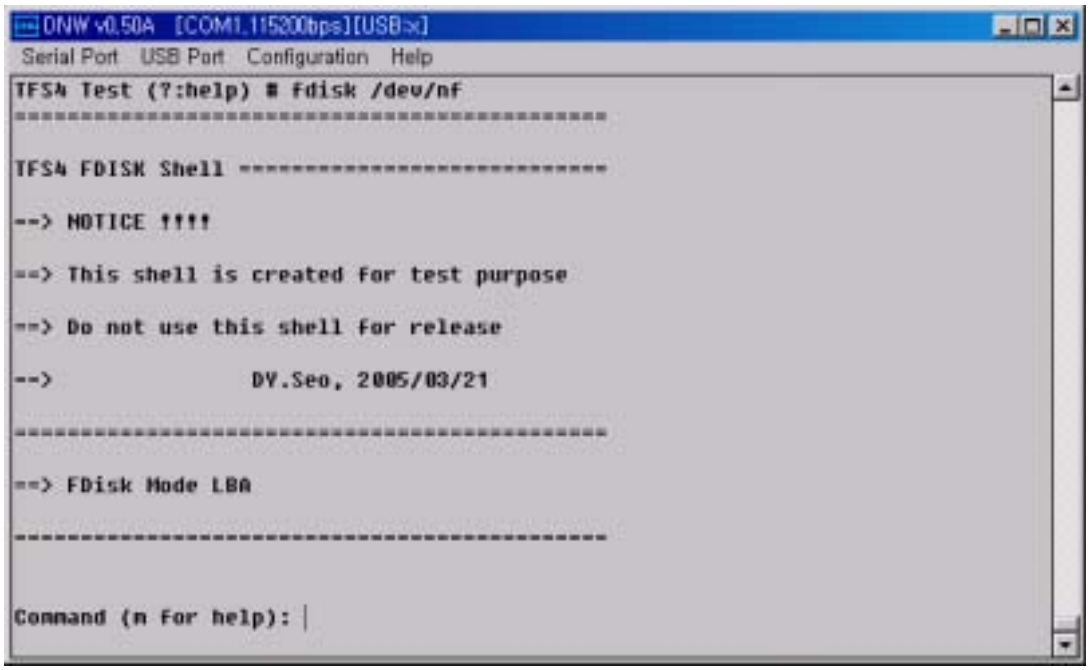


Figure 5-9. Perform tfs4_fdisk

2. void tfs4_do_fdisk(t_int8 *psDevice) is executed in test shell

3. Enter “m” to see the fdisk command.

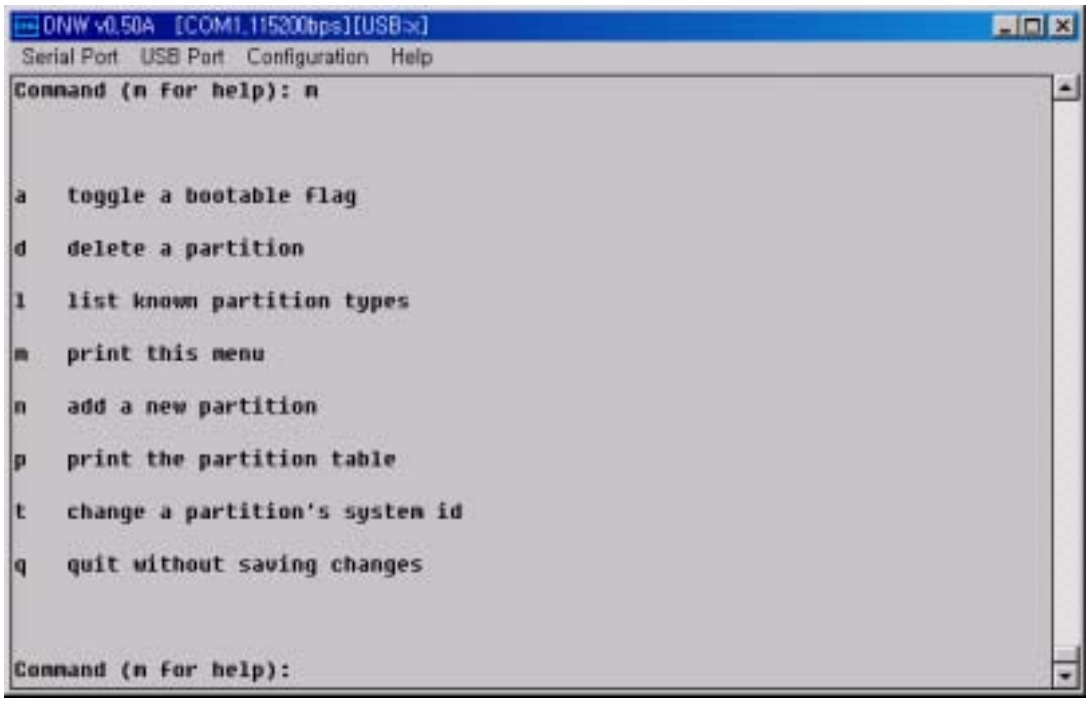


Figure 5-10. fdisk Commands

4. The following picture shows that a partition is created by using fdisk commands.

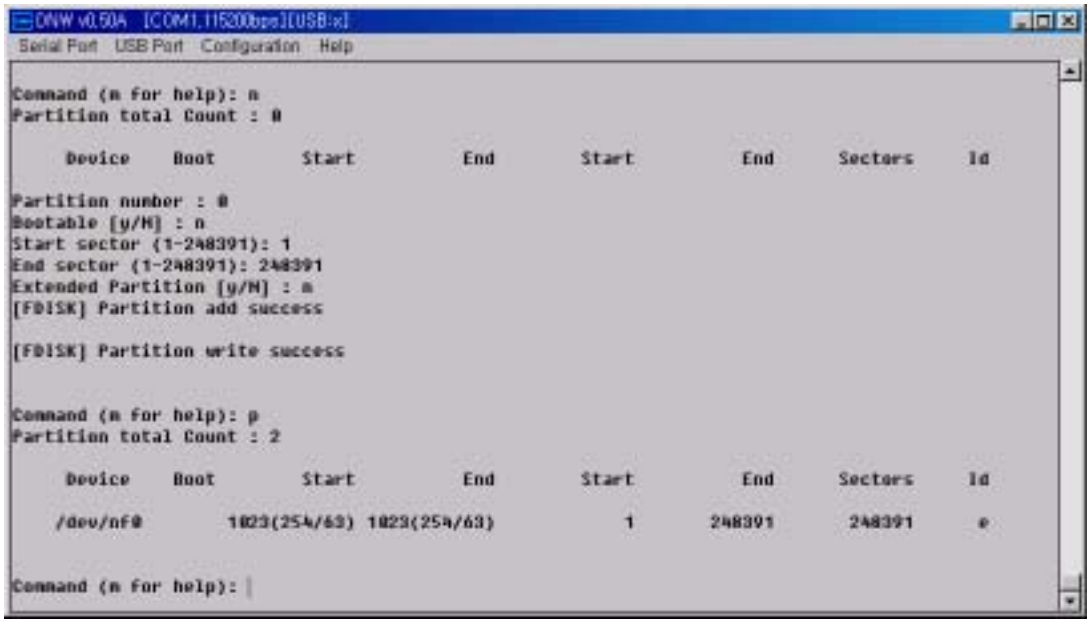


Figure 5-11. See the Created Partition

After the partition is created, enter “q” to quit the fdisk setting.

5.3.4. Format a volume

TFS4_format should be performed to format TFS4

Enter “tfs4_format {Device} {FilesystemType} {ClusterSize}” on the test shell.

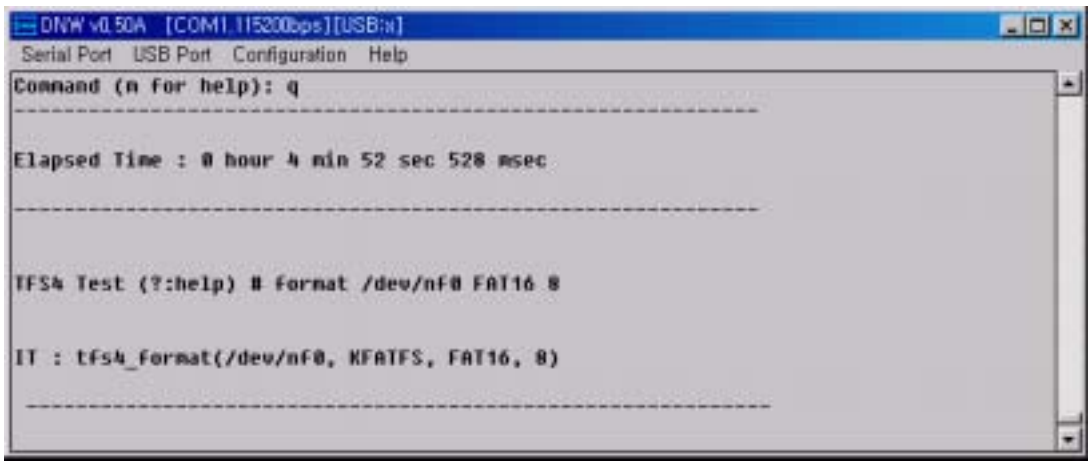


Figure 5-12. Perform tfs4_format

tfs4_format is executed in test shell and you can see the printed message that writing FAT table is finished on NAND device.

5.3.5. Mount a volume

TFS4_mount should be done for TFS4 to use NAND device. If tfs4_format is not performed, tfs4_mount returns fail.

Enter “tfs4_mount {LogicalDevice} {TargetVolume} {Filesystem} {flag}” on the test shell.

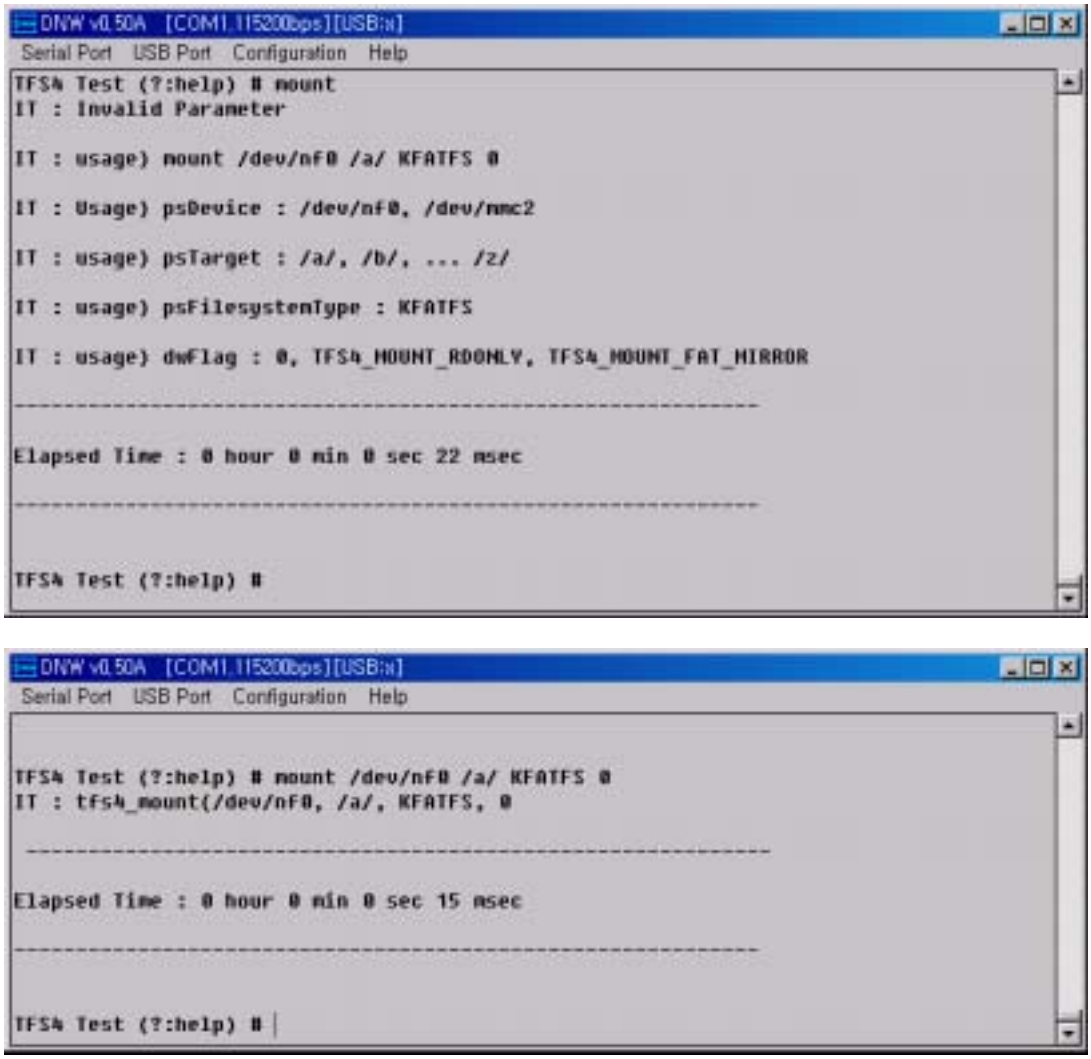


Figure 5-13. Perform tfs4_mount

tfs4_mount is executed in test shell and you can see the printed message on the test shell.

5.3.6. Case test & Stress test

There is a case test command that performs tfs4 open, read, or write function test over and over. Case test is a collection of commands that performs basic functionalities of TFS4.

If case test is performed successfully, TFS4 is normally ported and running on target with no errors. Case test takes about 10 minutes. If the error occurs, the test shell shows the error messages and stops running.

The following screen shows that case test is executed.

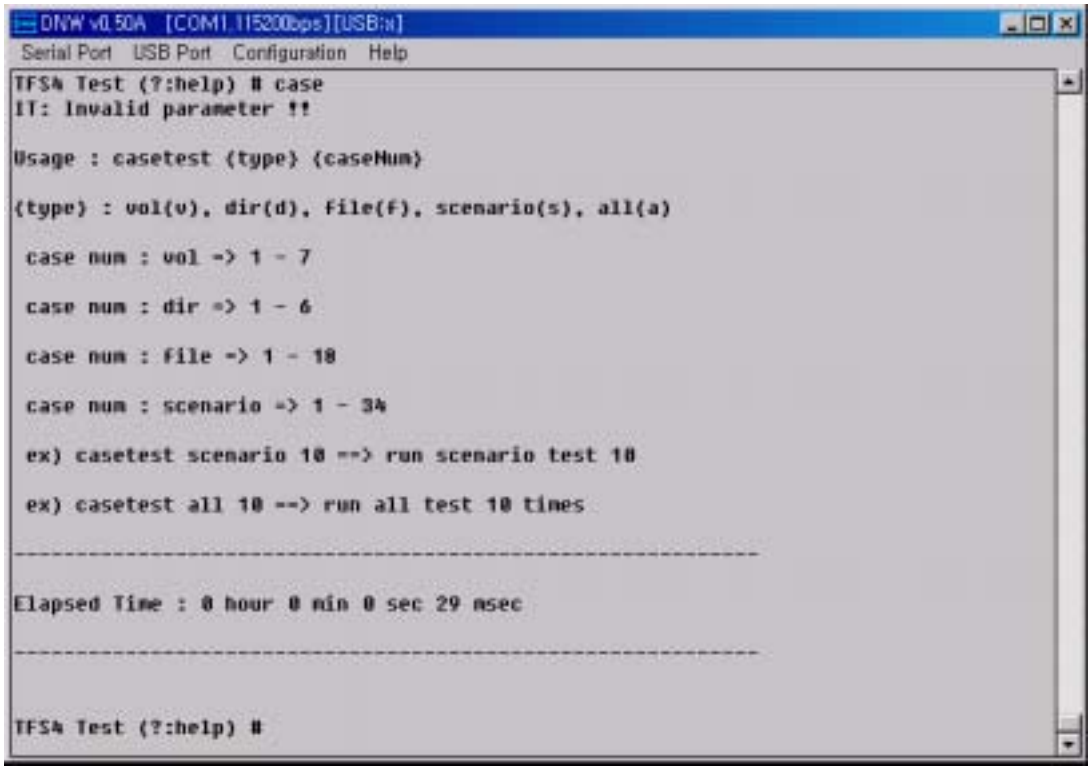


Figure 5-14. Perform a Case Test

A stress test is a random test of TFS4 for file and directory.

Enter “stress {test count}” on the test shell.

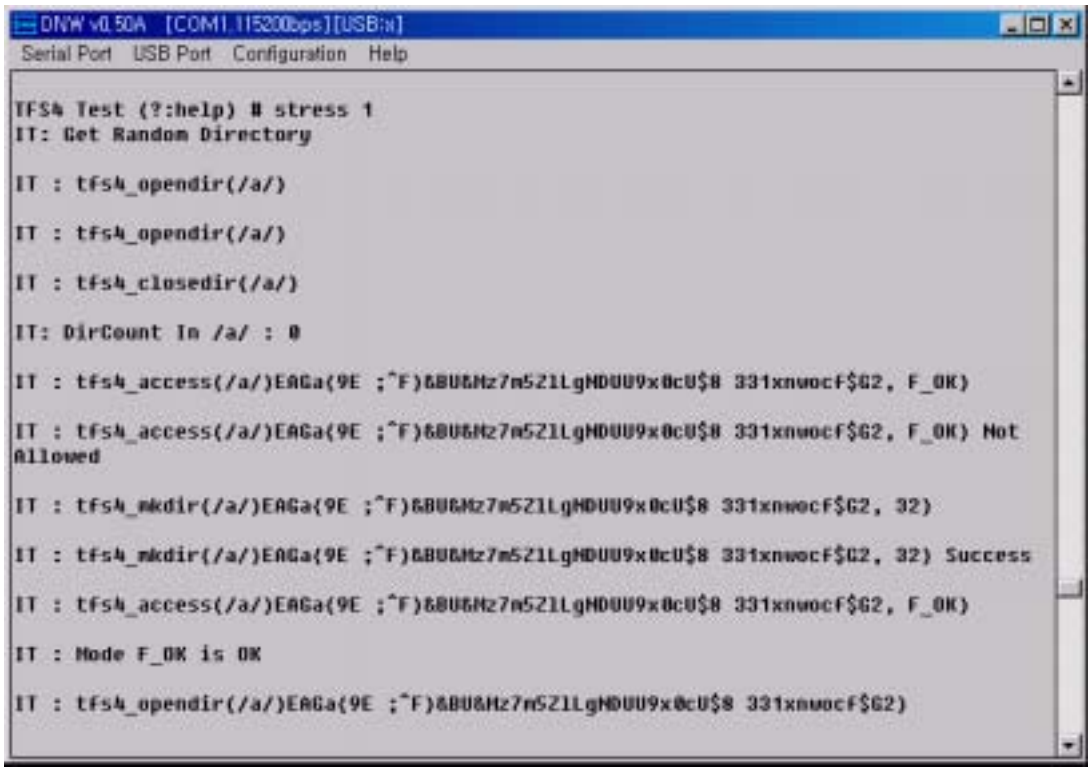


Figure 5-15. Perform a Stress Test

Appendix

Appendix covers the useful matters when you follow the TFS4 porting procedure. Also, it may help application programmers to develop an application based on TFS4. Appendix includes the seven sections as follows.

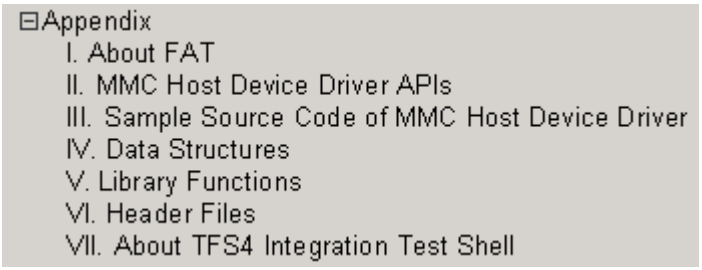


Figure 5-16. Contents of Appendix

You can see the above sections and go to the interested one.

I. About FAT

Following explains the overview, architecture and brief features of FAT.

- Overview

FAT is an abbreviation of File Allocate Table. This is a place where the location information of clusters[1] is stored.

TFS4 is compatible with FAT. Thus, the basic architecture of FAT is similar to that of TFS4. Following explains the architecture of FAT to help you understand the general architecture of TFS4.

Volume is a part of one physical disk. For example, it can be a “c drive” or “d drive” of your computer. A filesystem is used after formatted as one filesystem for one volume. At the space that is assigned as a volume, filesystem uses the space from the first sector to the last sector.

Following shows how FAT filesystem uses the first sector to the last sector, according to the FAT32[2] standard.
This is the basic structure.

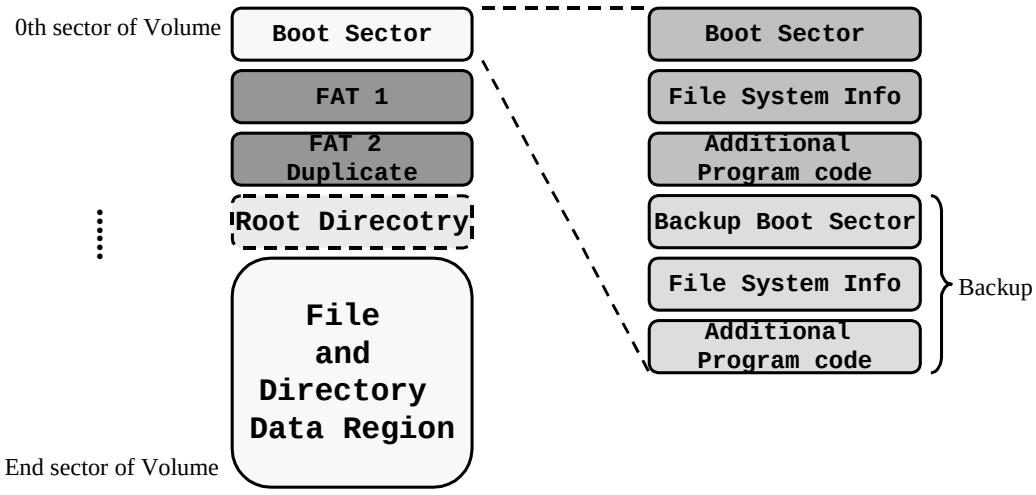


Figure 5-17. The Organization of FAT filesystem on Volume

FAT32 consists of following 4 regions.
- Reserved Region: Boot Sector and Additional block (FAT32 specification)
- FAT Region
- Root Directory Region
- File and Directory Data Region

Following explains each region in detail.

[1] Cluster is a logical unit for storing files into HDD.
[2] FAT32 holds a cluster with the minimum size 1KB to the maximum size 4KB.

- Reserved Region

First region, Reserved Region is an additional block that is only used in a Boot Sector and FAT32. It is composed of Filesystem Information Block, Additional program code block, and Backup space. This document introduces the information about Boot Sector.

Note

For more information about Filesystem Information Block, refer to the ***Microsoft Extensible Firmware Initiative FAT32 File System Specification***, Microsoft Corporation, Version 1.03, December 6, 2000.

Boot Sector is composed of 512 bytes. These 512 bytes are classified as five, which is as follows.

Byte Offset	Field Length	Field Name
0x00	3 bytes	Jump Instruction
0x03	LONGLONG	OEM ID
0x0B	53 bytes	BPB
0x40	26 bytes	Extended BPB
0x5A	420bytes	Bootstrap Code
0x01FE	WORD	End of Sector Marker

Figure 5-18. Boot Sector Structure

Jump Instruction, OEM ID, and Bootstrap Code are the codes that are used when a volume is able to boot. End of Sector Marker is a unique feature of FAT, which can confirm the last part of one sector. To access a volume, filesystem uses the record of Bios Parameter Block (BPB) as a standard. BPB has a standard value that fills a volume.

Followings are the example for the value:
Byte numbers of one sector, sector numbers that are allocated to a Reserved Region, specific value for Filesystem type(FAT16/32), Media descriptor, etc.

BPB uses total 79 bytes. Following shows the detailed structure.

Byte Offset	Field Length	Sample Value	Field Name
0x0B	WORD	0x0002	Byte Per Sector
0x0D	BYTE	0x02	**Sectors Per Cluster**
0x0E	WORD	0x2000	Reserved Sectors
0x10	BYTE	0x02	Number of FATs
0x11	WORD	0x0000	Root Entries (FAT12/FAT16 only)
0x13	WORD	0x0000	Small Sectors (FAT12/FAT16 only)
0x15	BYTE	0xF8	Media Descriptor
0x16	WORD	0x0000	Sectors Per FAT (FAT12/FAT16 only)
0x18	WORD	0x3F00	Sectors Per Track
0x1A	WORD	0xFF00	Number of Heads
0x1C	DWORD	0x00000000	Hidded Sectors
0x20	DWORD	0x00F00300	Large Sectors
0x24	DWORD	0xE9030000	Sectors Per FAT (FAT32 only)
0x28	WORD	0x0000	Extended Flags (FAT32 only)
0x2A	WORD	0x0000	Filesystem Version (FAT32 only)
0x2C	DWORD	0x02000000	Root Cluster Number (FAT32 only)
0x30	WORD	0x0100	FSInfo Sector Number (FAT32 only)
0x32	WORD	0x0600	Backup Boot Sector (FAT32 only)
0x36	12 bytes	All zero	Reserved (FAT32 only)
0x40	BYTE	0x00	Physical Drive Number
0x41	BYTE	0x01	Reserved
0x42	BYTE	0x29	Extended Boot Signature
0x43	DWORD	0xE17B9822	Volume Serial Number
0x47	11 bytes	“NO NAME”	Volume Label

Figure 5-19. BPB Structure

BPB information is used for acquiring the specific values to create/delete/change files or directories. For example, to crate one directory, following information is needed;
the information about FAT Region and Root directory
the information for the first Data Sector.

To organize this information, each field of BPB is used.

- FAT Region

The number of sectors that can be allocated to the FAT starting sector and one FAT is obtained through BPB. Generally, FAT is used with a mirror. Each entry of FAT corresponds to the available cluster numbers as 1:1. If a FAT type is FAT16, the entry uses 16-bit unit. Also, if a FAT type is FAT32, the entry uses 32-bit unit.

Each entry contains specific values, which can be classified as follows:
Reserved Entry Value
Media Descriptor:
-. FAT16: 0xFFF8
-. FAT32: 0x0FFFFFF8
EOC Mark:
-. FAT16: 0xFFFF
-. FAT32: 0x0FFFFFFF
Special Entry Value
Bad Cluster
-. FAT16: 0xFFF7
-. FAT32: 0x0FFFFFF7
Free Cluster
-. FAT16: 0x0000
-. FAT32: 0x00000000
Normal Entry Value
: Generally, each entry has an EOC value that notifies the last of a cluster chain. Or, it has a cluster number of the next chain.

- Root Directory Region

Root Directory Region only exists in FAT16. At FAT32, a Root Directory is also used after allocated with a cluster number. Root Directory Region exists before the starting point of the first data sector.

It can access from the starting sector number to the last sector number. The last sector number is calculated by the number of directory entries that Root Directory of BPB can possess.

For more information for directory entries, refer to the next contents.

- File and Directory Data Region

Generally, a file has a data with a byte unit. However, directory has a data with a 32 bytes unit, a directory entry, to indicate lower directories and files.
The size of every file and directory can be changed. In other words, the number of clusters can be increased or decreased. At this point, FAT Region cluster chain is formed.

For example, this is a file named “File.txt.” Following shows how to form a cluster chain in a FAT Region.

```
            00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000h : F8 FF FF FF FF FF FF FF FF FF FF 0F 04 00 00 00
00000010h : FF FF FF 0F 06 00 00 00 07 00 00 00 08 00 00 00
00000020h : 09 00 00 00 0A 00 00 00 FF FF FF 0F 00 00 00 00
                              ...
000001F0h : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 5-20. Cluster chain on FAT

This picture shows the first sector of a real FAT Region. The red part is a cluster chain that is allocated to a “File.txt” to explain an example.

Next picture explains the procedure of finding out a cluster chain.

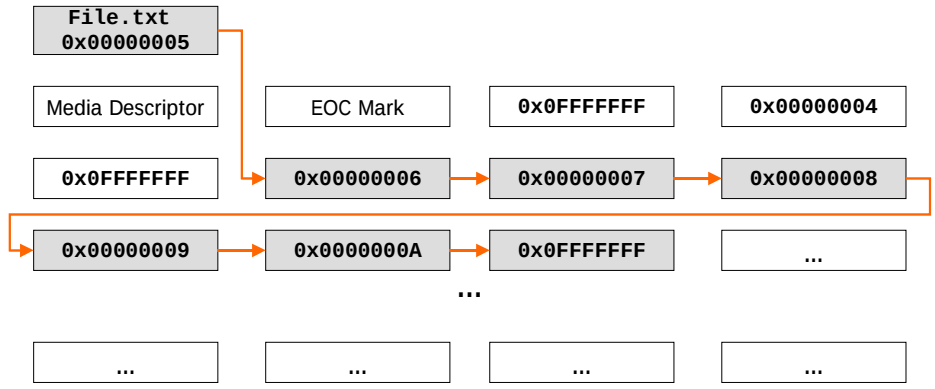


Figure 5-21. Cluster chain of the File.txt file

① “File.txt” starts from the 5th cluster. You can know this through a field that represents a first cluster number of a directory entry in “File.txt.”
② The FAT entry that corresponds to the 5th cluster in 1:1 at FAT region, is recorded as 0x00000006.
③ Find a FAT entry that meets 1:1 to the value from a FAT entry.
④ If the read FAT entry value is EOC value, it ends. If not, the ③ is repeated.

This procedure is applied to both a file and directory.

Directory entry includes the information for files or directories that belong to the current directory. The type of directory entry is classified as a short directory entry and long directory entry. According to the length of a file name, a short name has a short directory entry and a long name has a combination of a short directory entry and long directory entry. Following picture displays the structure of a short directory entry and long directory entry.

Byte Offset	Field Length	Field Name
0x00	**8 bytes**	**Filename**
0x08	**3 bytes**	**Filename extension**
0x0B	BYTE	Attribute byte (Bit coded)
0x0C	BYTE	Reserved for use by Windows NT
0x0D	BYTE	Millisecond stamp at file creation time
0x0E	WORD	Time file was created
0x10	WORD	Date file was created
0x0E	WORD	Last Access date
0x10	WORD	High word of the starting cluster number
0x0E	WORD	Time of last write
0x10	WORD	Date of last write
0x0E	WORD	Low word of the starting cluster number
0x10	DWORD	File Size as 32-bit DWORD

Figure 5-22. Short Directory Entry Structure

Byte Offset	Field Length	Field Name
0x00	BYTE	**Order of directory entry**
0x01	**10 bytes**	**Character 1~5 of the long-name**
0x0B	BYTE	Attribute byte (Bit coded)
0x0C	BYTE	Type
0x0D	BYTE	Checksum of name in short dentry
0x0E	**12 bytes**	**Character 6~11 of the long-name**
0x1A	WORD	Low word of the starting cluster number
0x1C	DWORD	**Character 12~13 of the long-name**

Figure 5-23. Long Directory Entry Structure

Note

For more information about handling the long name in a file name and each directory entry, refer to the Microsoft Extensible Firmware Initiative FAT32 File System Specification, Microsoft Corporation, Version 1.03, December 6, 2000, Long Filename Specification, Microsoft Corporation, Version 0.5, December 4, 1992.

II. MMC (or HSMMC) Host Device Driver APIs

This section describes MMC(or HSMMC) host device driver API adapted for TFS4 filesystem. The APIs are listed from the next page. The APIs should be provided when you writes a MMC(or HSMMC) host device driver.

This is the sample APIs of MMC(or HSMMC) host device driver.

```
t_int32     mmc_init_driver   (void)
t_int32     mmc_is_ready      (void)
t_int32     mmc_read    (t_uint8       *pBuf,       t_uint32
uiStartSector, t_uint32 uiNumSectors)
t_int32     mmc_write   (t_uint8       *pBuf,       t_uint32
uiStartSector, t_uint32 uiNumSectors)
void  mmc_get_stat      (t_mmc_info* pBuf)
```

mmc_init_driver

DESCRIPTION

This function initializes MMC(or HSMMC).

SYNTAX

```
t_int32 mmc_init_driver(void)
```

PARAMETERS

Parameter	Description
void	

RETURN VALUE

Return Value	Description
0	Success
Less than 0	Failure

REMARKS

This function initializes and enables normal I/O of MMC(or HSMMC). It returns a negative number on failure. If this function is a success, it makes TRUE when mmc_is_ready is called.

EXCEPTIONS

EXAMPLE

```
mmc_init_driver();
```

SEE ALSO

mmc_is_ready

mmc_is_ready

DESCRIPTION

This function checks whether MMC(or HSMMC) initialization is fail or success.

SYNTAX

```
t_int32 mmc_is_ready(void)
```

PARAMETERS

Parameter	Description
void	

RETURN VALUE

Return Value	Description
1	MMC is initialized successfully
0	MMC initialization failed.

REMARKS

This function confirms MMC(or HSMMC) initialization. This function can not be used to know if MMC(or HSMMC) is inserted or ejected.

EXCEPTIONS

EXAMPLE

```
mmc_is_ready();
```

SEE ALSO

```
mmc_init_driver
```

mmc_read

DESCRIPTION

This function reads a data sector from MMC(or HSMMC) .

SYNTAX

```
t_int32 mmc_read(t_uint8 *pBuf, t_uint32 uiStartSector, t_uint32
uiNumSectors)
```

PARAMETERS

Parameter	Description
pBuf	Buffer pointer to store data into
uiStartSector	Starting sector number for data read operation
uiNumSectors	The number of sectors to read

RETURN VALUE

Return Value	Description
0	Success
Negative	Failure

REMARKS

This function has to send the requested data from TFS4 File System to the buffer. It performs I/O whose transfer unit is a sector (512 bytes), takes the arguments, the starting sector number and the number of sector to read, and copies data. Filesystem guarantees enough memory size for pBuf by the parameter and does not support alignment of buffer pointer. Thus, the alignment should be handled in the mmc_read function, if necessary.

EXCEPTIONS

EXAMPLE

```
mmc_read(pBuf, 0, 1024);
```

SEE ALSO

mmc_write

mmc_write

DESCRIPTION

This function writes a sector of data on MMC(or HSMMC).

SYNTAX

```
t_int32  mmc_write(t_uint8  *pBuf,  t_uint32  uiStartSector,
t_uint32 uiNumSectors)
```

PARAMETERS

Parameter	Description
pBuf	Starting pointer of buffer storing data to write
uiStartSector	Starting sector to write data
uiNumSectors	The number of sector to write data

RETURN VALUE

Return Value	Description
0	Success
Less than 0	Failure

REMARKS

This function has to write the requested data from TFS4 File System on a MMC(or HSMMC) sector. It performs I/O whose transfer unit is a sector (512 byte), takes the arguments, the starting sector number and the number of sector to write, and writes data on MMC. It does not guarantee alignment of buffer pointer. Thus, the alignment should be handled in the mmc_write function, if necessary.

EXCEPTIONS

EXAMPLE

```
mmc_write(pBuf, 0, 1024);
```

SEE ALSO

mmc_read

mmc_get_stat

DESCRIPTION

This function retrieves the information of MMC(or HSMMC) device.

SYNTAX

```
void mmc_get_stat(t_mmc_info* pBuf)
```

PARAMETERS

Parameter	Description
pBuf	Buffer pointer where the information of t_mmc_info type is stored.

RETURN VALUE

Return Value	Description
void	

REMARKS

TFS4 filesystem has to send the physical information and other additional information to format MMC(or HSMMC). In case that the device driver gets a request from TFS4 file system, this function has to read the MMC(or HSMMC) register value, process, and pass it over as the defined in t_mmc_info type.

bSectorPerTrack, bTracks, and wCylinders are geometric values and do not exist in MMC(or HSMMC). They are calculated by using _get_geometrics() in the mmc_command.c file. Refer to sample MMC host device driver.

EXCEPTIONS

EXAMPLE

```
mmc_get_stat(pBuf);
```

SEE ALSO

III. Data Structures

Data structure for application programmer is defined in tfs4_types.h. You can refer to tfs4_types_internal.h for TFS4 Porting.

The following is the data types in tfs4_types.h for application programmer.

Table 23. Data Types of TFS4

```
typedef unsigned char         t_uint8;
typedef char                  t_int8;
typedef short int             t_int16;
typedef unsigned short int    t_uint16;
typedef int                   t_int32;
typedef unsigned int          t_uint32;

typedef t_int16         ssize_t;
typedef t_uint32        mode_t;
typedef t_int32         off_t;

typedef struct
{
    t_uint8    sDir_Name[11];
    t_uint8    cDir_Attr;
    t_uint8    cDir_NTRes;
    t_uint8    cDir_CrtTimeTenth;
    t_uint16   wDir_CrtTime;
    t_uint16   wDir_CrtDate;
    t_uint16   wDir_LstAccDate;
    t_uint16   wDir_FstClusHi;
    t_uint16   wDir_WrtTime;
    t_uint16   wDir_WrtDate;
    t_uint16   wDir_FstClusLo;
    t_uint32   dwDir_FileSize;
} t_dir_entry;

typedef struct
{
    t_uint32   st_mode;    /* file mode */
    t_uint32   st_ino;     /* file serial number */
    t_int16    st_dev;     /* ID of device containing this file
    */
    t_int16    st_dummy;   /* dummy entry */
    t_uint32   st_size;    /* the file size in bytes */
    t_uint32   st_atime;   /* time of last access */
    t_uint32   st_mtime;   /* time of last data modification */
    t_uint32   st_ctime;   /* time of last status change */
} t_stat;

typedef struct
{
    t_int32 f_type;        /* type of filesystem */
    t_int32 f_bsize;         /* optimal transfer block size,
```

```
    cluster size*/
    t_int32 f_bsizebits;    /* block size in bits */
    t_int32 f_blocks;      /* total data blocks in file system,
    total cluster count */
    t_int32 f_bfree;       /* free blocks in fs, free cluster
    count */
    t_int32 f_bavail;      /* free blocks avail to non-superuser,
    equal to f_bfree */
    t_int32 f_files;       /* total file nodes in file system
    */
    t_int32 f_ffree;       /* free file nodes in fs */
    t_int32 f_fsid;        /* file system id */
    t_int32 f_maxfilesize;  /* maximum file size */
    t_int16 f_namelen;     /* maximum length of filenames */
    t_uint8 f_dummy[2];    /* dummy for alignment */
} t_statfs;

typedef struct
{
    t_int8 d_name[512];
} t_dirent;

// directory stream class
typedef struct
{
    t_int32    fd;        /* fd for the open directory */
    t_dirent   dent;      /* directory entry buffer */
    t_int16    offset;    /* current offset */
    t_int16    index;     /* internal data */
    t_uint8    dummy[2];   /* for alignment */
} t_DIR;
```

IV. Library Functions

For information on TFS4 library functions, refer to TFS4 programmer’s guide.

V. Header Files

This section describes the TFS4 header files. The TFS4 header files can be classified as follows:

- Header files for TFS4 porting
- Header files that an application programmer has to include

Here we assume that TFS4 is ported to OS completely and an application programmer uses a TFS4 in a library; an application programmer may use the TFS4 API using the structure. The structure is in the TFS4 header file.

This is the list of header files for application programmer. They can include it to their application.

- tfs4_api.h
- tfs4_config_const.h
- t fs4_config.h
- tfs4_errno.h
- tfs4_global.h
- tfs4_types.h

VI. Error Codes

Following represents the TFS4 error codes and description.

Table 24. Error Codes List

Error Code	Error Number	Error Description
TFS4_OK	-0x00000000	not error
TFS4_EPROG	-0x00000001	programming error
TFS4_ENOMEDIA	-0x00000002	there is no media
TFS4_EMEDIAFAIL	-0x00000003	media is damaged
TFS4_ENOMEM	-0x00000004	no memory
TFS4_EIO	-0x00000005	I/O error
TFS4_EINVALID	-0x00000006	Invalid argument
TFS4_ENOSUPPORT	-0x00000007	Unsupported operation request
TFS4_EPANIC	-0x00000008	panic, un-recoverable error
TFS4_ENODEV	-0x00000009	no such device error
TFS4_EBUSY	-0x0000000A	the device is busy
TFS4_EINVALIDPATH	-0x0000000B	invalid PATH
TFS4_EBADF	-0x0000000C	bad file descriptor
TFS4_EFAULT	-0x0000000D	invalid path pointer
TFS4_EEXIST	-0x0000000E	file or directory already exists
TFS4_ENOENT	-0x0000000F	no such file or directory
TFS4_EACCESS	-0x00000010	invalid access
TFS4_EINVAL	-0x00000011	invalid rename path
TFS4_ENAMETOOLONG	-0x00000012	too long path
TFS4_EISDIR	-0x00000013	invalid operation try for a directory
TFS4_EISFILE	-0x00000014	invalid operation try for a file
TFS4_EEJECT	-0x00000015	media is ejected
TFS4_ENOTDIR	-0x00000016	not directory
TFS4_ENFILE	-0x00000017	too many files are currently open in the system
TFS4_EROFS	-0x00000018	write access to read only volume
TFS4_ENOTEMPTY	-0x00000019	not empty directory
TFS4_EXDEV	-0x0000001A	volume is different
TFS4_ENOSPC	-0x0000001B	no room at the device
TFS4_ELOGDIR	-0x0000001C	directory with the same name to the log file exists → user should delete it and try again
TFS4_ELOG_CREAT	-0x0000001D	failed to create log file
TFS4_ELOG_RECOVERY	-0x0000001E	
TFS4_ETOOLONG	-0x0000001F	file/directory name is too long
TFS4_EFAT	-0x00000020	problem in FAT chain
TFS4_EIO_SOFT	-0x00000021	FTL ECC error
TFS4_EPDEV_INIT	-0x00000022	XSR STL_Init() error
TFS4_EPDEV_OPEN	-0x00000023	XSR STL_Open() error
TFS4_EINIT_ALREADY	-0x00000024	TFS4 already initialized
TFS4_EINIT	-0x00000025	TFS4 is not initialized yet.

VII. About TFS4 Integration Test Shell

TFS4 provides a test shell, so you can perform a TFS4 integration test on your host while TFS4 is running on your target. The following picture shows the directory that includes a shell source file, tfs4_integration_test.c.

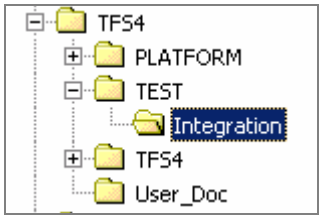


Figure 5-24. Directory Path of tfs4_integration_test.c

The directory path of the test shell is “C:\TFS4\TEST\INTEGRATION” The directory includes two file: a tfs4_integration_test.c and tfs4_integration_test.h

To use this shell, you first have to set a shell memory to use. You create a task and make tf4_main() is called from the task.

The stack and heap size for the task is as follows.

Table 25. Stack and Heap size

Memory	The least Memory Usage
Stack	60 KB
Heap	4 MB

As you see, a heap size is a little large, because a high capacity of buffer is allocated and tested for write operation.

This shell has a lot of commands to test the basic features of TFS4. Use a “?” command to see a shell help menu.

You can configure the shell memory usage and message print setting through UART in the tfs4_integration_test.h file.

This is the screen that TFS4 is initialized.

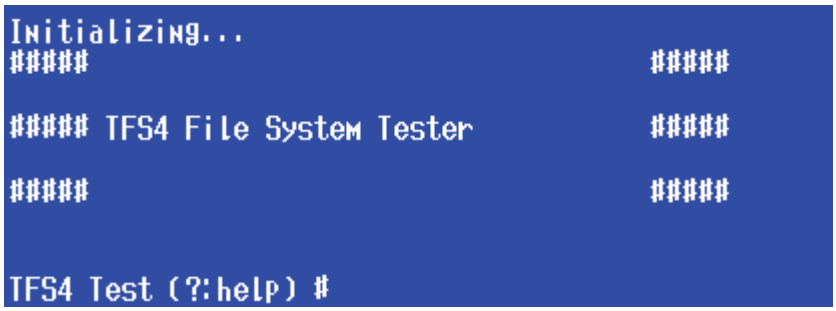


Figure 5-25. Test Shell Screen

To see a help menu, input “?”, “h”, or “help” on your test shell.

It is created for TFS4 API test, and you can execute a simple TFS4 API.

For example, if you try to initialize TFS4, you can enter “tfs4_init” on the shell. For TFS4 format, you can enter “tfs4_format /dev/nf0 fat16 8.”; refer to the help menu for command input order.

Most API can be used as listed in TFS4 programmer’s guide. But, you cannot test a tfs4_read or tfs4_write, because the buffer cannot be specified for those operations.

If you enter “tfs4_read 0 1024”, the test shell reads 1024 bytes of data from file which fd is 0, and prints it.

The following table lists the commands in the test shell.

Table 26. Test Shell Command

Command	Description
quit, q, exit	Finish the test
?, help, h	Print Help
tfs4_init	Initialize TFS4 File System
tfs4_termiante	Terminate TFS4 File System
tfs4_pdev_reg	Register a physical device
tfs4_pdev_unreg	Un-register a physical device
tfs4_mount	Mount Volume Usage) tfs4_mount {Logical Device Name} {Target} {Filesystem} {flag}
tfs4_umount	Un-Mount Volume Usage) tfs4_umount {VolumeName}
tfs4_umount2	Un-mount volume with flag Usage) tfs4_umount2 {VolumeName} {flag}
tfs4_vm_mount_mmc	Mount MMC Usage) tfs4_vm_mount_mmc
tfs4_statfs	get volume status

	Usage) tfs4_statfs {VolumeName}
tfs4_format	format file system Usage) tfs4_format {Device} {FilesystemType} {ClusterSize} ex) format /dev/nf0 FAT16 4",
tfs4_fdisk	fdisk utility Usage) fdisk {device} Ex) fdisk /dev/nf
tfs4_ioctl	Run tfs4_ioctl ex) tfs4_ioctl /dev/mmc enuIOCTL_MMC_GET_INFO
tfs4_opendir	open a directory Usage) tfs4_opendir {path}
tfs4_closedir	close a directory Usage) tfs4_opendir {path}
tfs4_mkdir	make a directory Usage) tfs4_mkdir {path} {mode}
tfs4_rmdir	remove a directory Usage) tfs4_rmdir {path}
tfs4_readdir	read directory entry Usage) tfs4_readdir {path}
tfs4_rewinddir	rewind directory entry Usage) tfs4_rewinddir {path}
print_open_dir	print open directory list
tfs4_open	open a file Usage) tfs4_open {path} {flag}
tfs4_close	close a file Usage) tfs4_close {fd}
tfs4_read	read file Usage) tfs4_read {fd} {byte}
tfs4_write	write to file Usage) tfs4_write {fd} {byte}
tfs4_create	create a file Usage) tfs4_create {path} {mode}
tfs4_rename	change file name Usage) tfs4_rename {oldpath} {newpath}
tfs4_truncate	truncate a file Usage) tfs4_truncate {path} {size}
tfs4_ftruncate:	truncate a file using FD Usage) tfs4_ftruncate {fd} {size}
tfs4_stat	get file stat Usage) tfs4_stat {path}
tfs4_fstat	get file stat using FD Usage) tfs4_fstat {fd}
tfs4_unlink	unlink file Usage) tfs4_unlink {path}
tfs4_feof	end of file check Usage) tfs4_feof {fd}
tfs4_ftell	get file pointer Usage) tfs4_ftell {fd}"
tfs4_lseek	set file pointer Usage) tfs4_lseek {fd} {offset} {whence}
tfs4_fsynk	sync file Usage) tfs4_fsync {fd}
tfs4_sync	sync device

	Usage) tfs4_sync {device name}
tfs4_access	file access check Usage) tfs4_access {path} {mode}
print_open_file	print open file name & FD, alias 'pof' Usage) pof
close_file	close a file, alias 'closefile' Usage) closefile {path}
read_file	read file, alias 'readfile' Usage) readfile {path} {byte}
write_file	write file, alias 'writefile' Usage) writefiled {path} {byte}
tfs4_backup	Make a copy of file system metadata. Usage) backup {volume} {rescue file path} {options} {buffer size}
tfs4_restore	Restore a file system to the old state when the rescue file had been created. Usage) restore {device name} {rescue file path} {options} {buffer size}
rs	read sector Usage) rs {device_name} {sec_no} device_name : /dev/nf, /def/mmc
stl_format	format stl
bml_format	format BML
casetest	case test usage) case {type} {caseNum}
stress	random stress test usage) stress {count}
dir	print dir and files usage) dir {path}
deltree	erase directory tree usage) deltree {path}
set_test_vol	set test volume usage) set_test_vol {volume_name} ex) set_test_vol /e/
set_rand_seed	set random seed usage) set_rand_seed {value}
perf	performance test (Sequential test) usage) perf {mode} {test_count} mode) tfs4_read, tfs4_write, tfs4_readwrite mode) xsr_read, xsr_write, xsr_readwrite, all tfs4_timer_expire() should be called once a 0.01 sec. by using the timer interrupt for performing it exactly.

VIII. Code Pages

Code Page & UNICODE

949 : KOREAN

437 : US

850 : Multilingual Latin I

852 : Multilingual Latin II
855 : Cyrillic
857 : Turkish
858 : Multilingual Latin I + Euro
862 : Hebrew
866 : Russian
874 : Thai
932 : Japanese Shift-JIS
936 : Simplified Chinese GBK
949 : Korean
950 : Traditional Chinese Big5
1258 : Vietnam <== does not support now.
1250 : Central Europe
1251 : Central Europe
1251 : Cyrillic
1252 : Latin I
1253 : Greek
1254 : Turkish
1255 : Hebrew
1256 : Arabic
1257 : Baltic
1258 : Vietnam
28591 : IS08859_1 Latin 1
28592 : IS08859_2 Latin 2
28593 : IS08859_3 Latin 3
28594 : IS08859_4 Baltic
28595 : IS08859_5 Cyrillic
28596 : IS08859_6 Arabic
28597 : IS08859_7 Greek
28598 : IS08859_8 Hebrew
28599 : IS08859_9 Turkish
28605 : IS08859_15 Latin 9

reference page :
http://www.microsoft.com/globaldev/reference/cphome.mspx

YOU CAN NOT USE CODEPAGE 0 and 1, which are undefined code page number.

Language	Locale	ANSI CodePage	OEM CodePage
Afrikaans	Afrikaans	1252	850
Albanian	Albanian	1250	852
Arabic	Arabic (Algeria)	1256	720
Arabic	Arabic (Bahrain)	1256	720
Arabic	Arabic (Egypt)	1256	720
Arabic	Arabic (Iraq)	1256	720
Arabic	Arabic (Jordan)	1256	720
Arabic	Arabic (Kuwait)	1256	720
Arabic	Arabic (Lebanon)	1256	720

Arabic	Arabic (Libya)	1256	720
Arabic	Arabic (Morocco)	1256	720
Arabic	Arabic (Oman)	1256	720
Arabic	Arabic (Qatar)	1256	720
Arabic	Arabic (Saudi Arabia)	1256	720
Arabic	Arabic (Syria)	1256	720
Arabic	Arabic (Tunisia)	1256	720
Arabic	Arabic (U.A.E.)	1256	720
Arabic	Arabic (Yemen)	1256	720
Armenian	Armenian	0	1
Azeri (Cyrillic)	Azeri (Cyrillic)	1251	866
Azeri (Latin)	Azeri (Latin)	1254	857
Basque	Basque	1252	850
Belarusian	Belarusian	1251	866
Bulgarian	Bulgarian	1251	866
Catalan	Catalan	1252	850
Chinese	Chinese (Hong Kong S.A.R.)	950	950
Chinese	Chinese (Macau S.A.R.)	950	950
Chinese	Chinese (PRC)	936	936
Chinese	Chinese (Singapore)	936	936
Chinese	Chinese (Taiwan)	950	950
Croatian	Croatian	1250	852
Czech	Czech	1250	852
Danish	Danish	1252	850
Divehi	Divehi	0	1
Dutch	Dutch (Belgium)	1252	850
Dutch	Dutch (Netherlands)	1252	850
English	English (Australia)	1252	850
English	English (Belize)	1252	850
English	English (Canada)	1252	850
English	English (Caribbean)	1252	850
English	English (Ireland)	1252	850
English	English (Jamaica)	1252	850
English	English (New Zealand)	1252	850
English	English (Philippines)	1252	437
English	English (South Africa)	1252	437
English	English (Trinidad)	1252	850
English	English (United Kingdom)	1252	850
English	English (United States)	1252	437
English	English (Zimbabwe)	1252	437
Estonian	Estonian	1257	775
Faroese	Faroese	1252	850
Farsi	Farsi	1256	720
Finnish	Finnish	1252	850

French	French (Belgium)	1252	850
French	French (Canada)	1252	850
French	French (France)	1252	850
French	French (Luxembourg)	1252	850
French	French (Monaco)	1252	850
French	French (Switzerland)	1252	850
FYRO	Macedonian FYRO Macedonian	1251	866
Galician	Galician	1252	850
Georgian	Georgian	0	1
German	German (Austria)	1252	850
German	German (Germany)	1252	850
German	German (Liechtenstein)	1252	850
German	German (Luxembourg)	1252	850
German	German (Switzerland)	1252	850
Greek	Greek	1253	737
Gujarati	Gujarati	0	1
Hebrew	Hebrew	1255	862
Hindi	Hindi	0	1
Hungarian	Hungarian	1250	852
Icelandic	Icelandic	1252	850
Indonesian	Indonesian	1252	850
Italian	Italian (Italy)	1252	850
Italian	Italian (Switzerland)	1252	850
Japanese	Japanese	932	932
Kannada	Kannada	0	1
Kazakh	Kazakh	1251	866
Konkani	Konkani	0	1
Korean	Korean	949	949
Kyrgyz	Kyrgyz (Cyrillic)	1251	866
Latvian	Latvian	1257	775
Lithuanian	Lithuanian	1257	775
Malay	Malay (Brunei Darussalam)	1252	850
Malay	Malay (Malaysia)	1252	850
Marathi	Marathi	0	1
Mongolian	Mongolian (Cyrillic)	1251	866
Norwegian (Bokmal)	Norwegian (Bokmal)	1252	850
Norwegian (Nynorsk)	Norwegian (Nynorsk)	1252	850
Polish	Polish	1250	852
Portuguese	Portuguese (Brazil)	1252	850
Portuguese	Portuguese (Portugal)	1252	850
Punjabi	Punjabi	0	1
Romanian	Romanian	1250	852
Russian	Russian	1251	866
Sanskrit	Sanskrit	0	1

Serbian (Cyrillic)	Serbian (Cyrillic)	1251	855
Serbian (Latin)	Serbian (Latin)	1250	852
Slovak	Slovak	1250	852
Slovenian	Slovenian	1250	852
Spanish	Spanish (Argentina)	1252	850
Spanish	Spanish (Bolivia)	1252	850
Spanish	Spanish (Chile)	1252	850
Spanish	Spanish (Colombia)	1252	850
Spanish	Spanish (Costa Rica)	1252	850
Spanish	Spanish (Dominican Republic)	1252	850
Spanish	Spanish (Ecuador)	1252	850
Spanish	Spanish (El Salvador)	1252	850
Spanish	Spanish (Guatemala)	1252	850
Spanish	Spanish (Honduras)	1252	850
Spanish	Spanish (International Sort)	1252	850
Spanish	Spanish (Mexico)	1252	850
Spanish	Spanish (Nicaragua)	1252	850
Spanish	Spanish (Panama)	1252	850
Spanish	Spanish (Paraguay)	1252	850
Spanish	Spanish (Peru)	1252	850
Spanish	Spanish (Puerto Rico)	1252	850
Spanish	Spanish (Traditional Sort)	1252	850
Spanish	Spanish (Uruguay)	1252	850
Spanish	Spanish (Venezuela)	1252	850
Swahili	Swahili	1252	437
Swedish	Swedish	1252	850
Swedish	Swedish (Finland)	1252	850
Syriac	Syriac	0	1
Tamil	Tamil	0	1
Tatar	Tatar	1251	866
Telugu	Telugu	0	1
Thai	Thai	874	874
Turkish	Turkish	1254	857
Ukrainian	Ukrainian	1251	866
Urdu	Urdu	1256	720
Uzbek (Cyrillic)	Uzbek (Cyrillic)	1251	866
Uzbek (Latin)	Uzbek (Latin)	1254	857
Vietnamese	Vietnamese	1258	1258

Glossary

Index