



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China





User's Manual

Micrium
For the Way Engineers Work

Micrium
1290 Weston Road, Suite 306
Weston, FL 33326
USA

www.micrium.com

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where Micrium Press is aware of a trademark claim, the product name appears in initial capital letters, in all capital letters, or in accordance with the vendor's capitalization preference. Readers should contact the appropriate companies for more complete information on trademarks and trademark registrations. All trademarks and registered trademarks in this book are the property of their respective holders.

Copyright © 2012 by Micrium except where noted otherwise. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher; with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

The programs and code examples in this book are presented for instructional value. The programs and examples have been carefully tested, but are not guaranteed to any particular purpose. The publisher does not offer any warranties and does not guarantee the accuracy, adequacy, or completeness of any information herein and is not responsible for any errors or omissions. The publisher assumes no liability for damages resulting from the use of the information in this book or for any infringement of the intellectual property rights of third parties that would result from the use of this information.

Table of Contents

Chapter 1	Introduction	15
1-1	µC/FS	15
1-2	Typical Usages	17
1-3	Why FAT?	17
1-4	Chapter Contents	18
Chapter 2	µC/FS Architecture	23
2-1	Architecture Components	25
2-1-1	Your Application	25
2-1-2	µC-LIB (Libraries)	25
2-1-3	POSIX API Layer	25
2-1-4	FS Layer	26
2-1-5	File System Driver Layer	27
2-1-6	Device Driver Layer	27
2-1-7	µC-CPU	28
2-1-8	RTOS Layer	28
Chapter 3	µC/FS Directories and Files	29
3-1	Application Code	32
3-2	CPU	34
3-3	Board Support Package (BSP)	35
3-4	µC/CPU, CPU Specific Source Code	36
3-5	µC/LIB, Portable Library Functions	38
3-6	µC/Clk, Time/Calendar Management	39
3-7	µC/CRC, Checksums and Error Correction Codes	41
3-8	µC/FS Platform-Independent Source Code	42
3-9	µC/FS FAT Filesystem Source Code	45
3-10	µC/FS Memory Device Drivers	46
3-11	µC/FS Platform-Specific Source Code	50

3-12	μC/FS OS Abstraction Layer	51
3-13	Summary	52
Chapter 4	Useful Information	59
4-1	Nomenclature	59
4-2	μC/FS Device and Volume Names	61
4-3	μC/FS File and Directory Names and Paths	62
4-4	μC/FS Name Lengths	64
4-5	Resource Usage	65
Chapter 5	Devices and Volumes	67
5-1	Device Operations	68
5-2	Using Devices	69
5-3	Using Removable Devices	71
5-4	Raw Device IO	72
5-5	Partitions	73
5-6	Volume Operations	76
5-7	Using Volumes	77
5-8	Using Volume Cache	79
5-8-1	Choosing Cache Parameters	81
5-8-2	Other Caching and Buffering Mechanisms	82
Chapter 6	Files	83
6-1	File Access Functions	84
6-1-1	Opening Files	85
6-1-2	Getting Information About a File	86
6-1-3	Configuring a File Buffer	87
6-1-4	File Error Functions	88
6-1-5	Atomic File Operations Using File Lock	88
6-2	Entry Access Functions	89
6-2-1	File and Directory Attributes	90
6-2-2	Creating New Files and Directories	91
6-2-3	Deleting Files and Directories	92
Chapter 7	Directories	93
7-1	Directory Access Functions	94

Chapter 8	POSIX API	95
8-1	Supported Functions	96
8-2	Working Directory Functions	97
8-3	File Access Functions	98
8-3-1	Opening, Reading and Writing Files	100
8-3-2	Getting or Setting the File Position	103
8-3-3	Configuring a File Buffer	104
8-3-4	Diagnosing a File Error	106
8-3-5	Atomic File Operations Using File Lock	106
8-4	Directory Access Functions	107
8-5	Entry Access Functions	109
Chapter 9	Device Drivers	111
9-1	Provided Device Drivers	112
9-1-1	Driver Characterization	113
9-2	Drivers comparison	115
Chapter 10	FAT File System	117
10-1	Why Embedded Systems Use FAT	117
10-2	Organization of a FAT Volume	118
10-2-1	Organization of Directories and Directory Entries	119
10-3	Organization of the File Allocation Table	120
10-3-1	FAT12 / FAT16 / FAT32	122
10-3-2	Short and Long File Names	123
10-4	Formatting	126
10-5	Sources of Corruption in FAT Volumes	127
10-6	Optional Journaling System	127
10-7	Licensing Issues	129
10-7-1	Licenses for Long File Names (LFNs)	129
10-7-2	Extended File Allocation Table (exFAT)	129
Chapter 11	RAM Disk Driver	131
11-1	Files and Directories	131
11-2	Using the RAM Disk Driver	132

Chapter 12	SD/MMC Drivers	135
12-1	Files and Directories	137
12-2	Using the SD/MMC CardMode Driver	138
12-2-1	SD/MMC CardMode Communication	141
12-2-2	SD/MMC CardMode Communication Debugging	144
12-2-3	SD/MMC CardMode BSP Overview	148
12-3	Using the SD/MMC SPI Driver	150
12-3-1	SD/MMC SPI Communication	153
12-3-2	SD/MMC SPI Communication Debugging	154
12-3-3	SD/MMC SPI BSP Overview	157
Chapter 13	NAND Flash Driver	159
13-1	Getting Started	160
13-2	Architecture Overview	167
13-3	NAND Translation Layer	168
13-3-1	Translation Layer Configuration	171
13-3-2	Translation Layer Source Files	178
13-4	Controller Layer	178
13-4-1	Generic Controller Layer Implementation	179
13-4-2	Part Layer	181
13-5	Board Support Package - Generic Controller	185
13-6	Board Support Package - Other Controllers	186
13-7	Performance Considerations	187
13-8	Development Guide	188
13-8-1	BSP Development Guide - Generic Controller	189
13-8-2	Generic Controller Extension Development Guide	191
13-8-3	ECC Module Development Guide	193
13-8-4	Controller Layer Development Guide	194
Chapter 14	NOR Flash Driver	199
14-1	Files and Directories	200
14-2	Driver and Device Characteristics	202
14-3	Using a Parallel NOR Device	204
14-3-1	Driver Architecture	208
14-3-2	Hardware	208
14-3-3	NOR BSP Overview	210

14-4	Using a Serial NOR Device	211
14-4-1	Hardware	212
14-4-2	NOR SPI BSP Overview	213
14-5	Physical-Layer Drivers	214
14-5-1	FSDev_NOR_AMD_1x08, FSDev_NOR_AMD_1x16	215
14-5-2	FSDev_NOR_Intel_1x16	215
14-5-3	FSDev_NOR_SST39	216
14-5-4	FSDev_NOR_STM25	216
14-5-5	FSDev_NOR_SST25	217
 Chapter 15	 MSC Driver	 219
15-1	Files and Directories	219
15-2	Using the MSC Driver	220
 Appendix A	 µC/FS API Reference	 223
A-1	General File System Functions	225
A-1-1	FS_DrvAdd()	226
A-1-2	FS_Init()	227
A-1-3	FS_VersionGet()	228
A-1-4	FS_WorkingDirGet()	229
A-1-5	FS_WorkingDirSet()	230
A-2	Posix API Functions	231
A-2-1	fs_asctime_r()	234
A-2-2	fs_chdir()	235
A-2-3	fs_clearerr()	236
A-2-4	fs_closedir()	237
A-2-5	fs_ctime_r()	238
A-2-6	fs_fclose()	239
A-2-7	fs_feof()	240
A-2-8	fs_ferror()	241
A-2-9	fs_fflush()	242
A-2-10	fs_fgetpos()	243
A-2-11	fs_flockfile()	244
A-2-12	fs_fopen()	245
A-2-13	fs_fread()	246
A-2-14	fs_fseek()	247
A-2-15	fs_fsetpos()	249

A-2-16	fs_ftell()	250
A-2-17	fs_ftruncate()	251
A-2-18	fs_ftrylockfile()	252
A-2-19	fs_funlockfile()	253
A-2-20	fs_fwrite()	254
A-2-21	fs_getcwd()	255
A-2-22	fs_localtime_r()	256
A-2-23	fs_mkdir()	257
A-2-24	fs_mktime()	258
A-2-25	fs_opendir()	259
A-2-26	fs_readdir_r()	260
A-2-27	fs_remove()	261
A-2-28	fs_rename()	263
A-2-29	fs_rewind()	265
A-2-30	fs_rmdir()	266
A-2-31	fs_setbuf()	267
A-2-32	fs_setvbuf()	268
A-3	Device Functions	270
A-3-1	FSDev_AccessLock()	273
A-3-2	FSDev_AccessUnlock()	274
A-3-3	FSDev_Close()	275
A-3-4	FSDev_GetDevName()	276
A-3-5	FSDev_GetDevCnt()	277
A-3-6	FSDev_GetDevCntMax()	278
A-3-7	FSDev_GetNbrPartitions()	279
A-3-8	FSDev_Invalidate()	280
A-3-9	FSDev_Open()	281
A-3-10	FSDev_PartitionAdd()	283
A-3-11	FSDev_PartitionFind()	284
A-3-12	FSDev_PartitionInit()	286
A-3-13	FSDev_Query()	287
A-3-14	FSDev_Rd()	288
A-3-15	FSDev_Refresh()	289
A-3-16	FSDev_Wr()	291
A-4	Directory Access Functions	292
A-4-1	FSDir_Close()	293
A-4-2	FSDir_IsOpen()	294
A-4-3	FSDir_Open()	295

A-4-4	FSDir_Rd()	296
A-5	Entry Access Functions	297
A-5-1	FSEntry_AttribSet()	298
A-5-2	FSEntry_Copy()	300
A-5-3	FSEntry_Create()	302
A-5-4	FSEntry_Del()	304
A-5-5	FSEntry_Query()	306
A-5-6	FSEntry_Rename()	307
A-5-7	FSEntry_TimeSet()	309
A-6	File Functions	311
A-6-1	FSFile_BufAssign()	313
A-6-2	FSFile_BufFlush()	315
A-6-3	FSFile_Close()	316
A-6-4	FSFile_ClrErr()	317
A-6-5	FSFile_IsEOF()	318
A-6-6	FSFile_IsErr()	319
A-6-7	FSFile_IsOpen()	320
A-6-8	FSFile_LockAccept()	321
A-6-9	FSFile_LockGet()	322
A-6-10	FSFile_LockSet()	323
A-6-11	FSFile_Open()	324
A-6-12	FSFile_PosGet()	326
A-6-13	FSFile_PosSet()	327
A-6-14	FSFile_Query()	329
A-6-15	FSFile_Rd()	330
A-6-16	FSFile_Truncate()	332
A-6-17	FSFile_Wr()	333
A-7	Volume Functions	335
A-7-1	FSVol_Close()	337
A-7-2	FSVol_Fmt()	338
A-7-3	FSVol_GetDfltVolName()	340
A-7-4	FSVol_GetVolCnt()	341
A-7-5	FSVol_GetVolCntMax()	342
A-7-6	FSVol_GetVolName()	343
A-7-7	FSVol_IsDflt()	344
A-7-8	FSVol_IsMounted()	345
A-7-9	FSVol_LabelGet()	346
A-7-10	FSVol_LabelSet()	348

A-7-11	FSVol_Open()	350
A-7-12	FSVol_Query()	352
A-7-13	FSVol_Rd()	353
A-7-14	FSVol_Wr()	355
A-8	Volume Cache Functions	356
A-8-1	FSVol_CacheAssign()	357
A-8-2	FSVol_CacheInvalidate ()	359
A-8-3	FSVol_CacheFlush ()	360
A-9	SD/MMC Driver Functions	361
A-9-1	FSDev_SD_xxx_QuerySD()	362
A-9-2	FSDev_SD_xxx_RdCID()	364
A-9-3	FSDev_SD_xxx_RdCSD()	366
A-10	NAND Driver Functions	368
A-10-1	FSDev_NAND_LowFmt()	369
A-10-2	FSDev_NAND_LowMount()	370
A-10-3	FSDev_NAND_LowUnmount()	372
A-11	NOR Driver Functions	373
A-11-1	FSDev_NOR_LowFmt()	374
A-11-2	FSDev_NOR_LowMount()	375
A-11-3	FSDev_NOR_LowUnmount()	376
A-11-4	FSDev_NOR_LowCompact()	377
A-11-5	FSDev_NOR_LowDefrag()	378
A-11-6	FSDev_NOR_PhyRd()	379
A-11-7	FSDev_NOR_PhyWr()	381
A-11-8	FSDev_NOR_PhyEraseBlk()	383
A-11-9	FSDev_NOR_PhyEraseChip()	385
A-12	FAT System Driver Functions	386
A-12-1	FS_FAT_JournalOpen()	387
A-12-2	FS_FAT_JournalClose()	388
A-12-3	FS_FAT_JournalStart()	389
A-12-4	FS_FAT_JournalStop()	390
A-12-5	FS_FAT_VolChk()	391
Appendix B	µC/FS Error Codes	393
B-1	System Error Codes	393
B-2	Buffer Error Codes	393
B-3	Cache Error Codes	394
B-4	Device Error Codes	394

B-5	Device Driver Error Codes	395
B-6	Directory Error Codes	395
B-7	ECC Error Codes	395
B-8	Entry Error Codes	395
B-9	File Error Codes	396
B-10	Name Error Codes	397
B-11	Partition Error Codes	397
B-12	Pools Error Codes	397
B-13	File System Error Codes	398
B-14	Volume Error Codes	398
B-15	OS Layer Error Codes	399
Appendix C	µC/FS Porting Manual	401
C-1	Date/Time Management	403
C-2	CPU Port	403
C-3	OS Kernel	404
C-4	Device Driver	412
C-4-1	NameGet()	414
C-4-2	Init()	415
C-4-3	Open()	416
C-4-4	Close()	418
C-4-5	Rd()	419
C-4-6	Wr()	421
C-4-7	Query()	423
C-4-8	IO_Ctrl()	424
C-5	SD/MMC Cardmode BSP	425
C-5-1	FSDev_SD_Card_BSP_Open()	429
C-5-2	FSDev_SD_Card_BSP_Lock/Unlock()	430
C-5-3	FSDev_SD_Card_BSP_CmdStart()	431
C-5-4	FSDev_SD_Card_BSP_CmdWaitEnd()	436
C-5-5	FSDev_SD_Card_BSP_CmdDataRd()	440
C-5-6	FSDev_SD_Card_BSP_CmdDataWr()	443
C-5-7	FSDev_SD_Card_BSP_GetBlkCntMax()	446
C-5-8	FSDev_SD_Card_BSP_GetBusWidthMax()	447
C-5-9	FSDev_SD_Card_BSP_SetBusWidth()	448
C-5-10	FSDev_SD_Card_BSP_SetClkFreq()	450
C-5-11	FSDev_SD_Card_BSP_SetTimeoutData()	451
C-5-12	FSDev_SD_Card_BSP_SetTimeoutResp()	452

C-6	SD/MMC SPI mode BSP	452
C-7	SPI BSP	453
C-7-1	Open()	457
C-7-2	Close()	459
C-7-3	Lock() / Unlock()	460
C-7-4	Rd()	461
C-7-5	Wr()	462
C-7-6	ChipSelEn() /ChipSelDis()	463
C-7-7	SetClkFreq()	464
C-8	NAND Flash Physical-Layer Driver	464
C-9	NOR Flash Physical-Layer Driver	464
C-9-1	Open()	467
C-9-2	Close()	468
C-9-3	Rd()	469
C-9-4	Wr()	470
C-9-5	EraseBlk()	471
C-9-6	IO_Ctrl()	472
C-10	NOR Flash BSP	473
C-10-1	FSDev_NOR_BSP_Open()	474
C-10-2	FSDev_NOR_BSP_Close()	475
C-10-3	FSDev_NOR_BSP_Rd_XX()	476
C-10-4	FSDev_NOR_BSP_RdWord_XX()	477
C-10-5	FSDev_NOR_BSP_WrWord_XX()	478
C-10-6	FSDev_NOR_BSP_WaitWhileBusy()	479
C-11	NOR Flash SPI BSP	480
Appendix D	µC/FS Types and Structures	481
D-1	FS_CFG	482
D-2	FS_DEV_INFO	484
D-3	FS_DEV_NOR_CFG	485
D-4	FS_DEV_RAM_CFG	488
D-5	FS_DIR_ENTRY (struct fs_dirent)	489
D-6	FS_ENTRY_INFO	490
D-7	FS_FAT_SYS_CFG	492
D-8	FS_PARTITION_ENTRY	494
D-9	FS_VOL_INFO	495

Appendix E	µC/FS Configuration	497
E-1	File System Configuration	498
E-2	Feature Inclusion Configuration	500
E-3	Name Restriction Configuration	503
E-4	Debug Configuration	504
E-5	Argument Checking Configuration	504
E-6	File System Counter Configuration	505
E-7	FAT Configuration	505
E-8	SD/MMC SPI Configuration	506
E-9	Trace Configuration	507
 Appendix F	 Shell Commands	 509
F-1	Files and Directories	510
F-2	Using the Shell Commands	511
F-3	Commands	514
F-3-1	fs_cat	515
F-3-2	fs_cd	516
F-3-3	fs_cp	518
F-3-4	fs_date	519
F-3-5	fs_df	520
F-3-6	fs_ls	521
F-3-7	fs_mkdir	522
F-3-8	fs_mkfs	523
F-3-9	fs_mount	524
F-3-10	fs_mv	525
F-3-11	fs_od	526
F-3-12	fs_pwd	527
F-3-13	fs_rm	528
F-3-14	fs_rmdir	529
F-3-15	fs_touch	530
F-3-16	fs_umount	531
F-3-17	fs_wc	532
F-4	Configuration	533

Appendix G	Bibliography	535
Appendix H	µC/FS Licensing Policy	537
H-1	µC/FS Licensing	537
H-1-1	µC/FS Source Code	537
H-1-2	µC/FS Maintenance Renewal	538
H-1-3	µC/FS Source Code Updates	538
H-1-4	µC/FS Support	538

Introduction

Files and directories are common abstractions, which we encounter daily when sending an e-mail attachment, downloading a new application or archiving old information. Those same abstractions may be leveraged in an embedded system for similar tasks or for unique ones. A device may serve web pages, play or record media (images, video or music) or log data. The file system software which performs such actions must meet the general expectations of an embedded environment—a limited code footprint, for instance—while still delivering good performance.

1-1 μ C/FS

μ C/FS is a compact, reliable, high-performance file system. It offers full-featured file and directory access with flexible device and volume management including support for partitions.

Source Code: μ C/FS is provided in ANSI-C source to licensees. The source code is written to an exacting coding standard that emphasizes cleanness and readability. Moreover, extensive comments pepper the code to elucidate its logic and describe global variables and functions. Where appropriate, the code directly references standards and supporting documents.

Device Drivers: Device drivers are available for most common media including SD/MMC cards, NAND flash, NOR flash. Each of these is written with a clear, layered structure so that it can easily be ported to your hardware. The device driver structure is simple—basically just initialization, read and write functions—so that μ C/FS can easily be ported to a new medium.

Devices and Volumes: Multiple media can be accessed simultaneously, including multiple instances of the same type of medium (since all drivers are re-entrant). DOS partitions are supported, so more than one volume can be located on a device. In addition, the logical device driver allows a single volume to span several (typically identical) devices, such as a bank of flash chips.

FAT: All standard FAT variants and features are supported including FAT12/FAT16/FAT32 and long file names, which encompasses Unicode file names. Files can be up to 4-GB and volumes up to 8-TB (the standard maximum). An optional journaling module provides total power fail-safety to the FAT system driver.

Application Programming Interface (API): μ C/FS provides two APIs for file and directory access. A proprietary API with parallel argument placement and meaningful return error codes is provided, with functions like `FSFile_Wr()`, `FSFile_Rd()` and `FSFile_PosSet()`. Alternatively, a standard POSIX-compatible API is provided, including functions like `fs_fwrite()`, `fs_fread()` and `fs_fsetpos()` that have the same arguments and return values as the POSIX functions `fwrite()`, `fread()` and `fsetpos()`.

Scalable: The memory footprint of μ C/FS can be adjusted at compile-time based on the features you need and the desired level of run-time argument checking. For applications with limited RAM, features such as cache and read/write buffering can be disabled; for applications with sufficient RAM, these features can be enabled in order to gain better performance.

Portable: μ C/FS was designed for resource-constrained embedded applications. Although μ C/FS can work on 8- and 16-bit processors, it will work best with 32- or 64-bit CPUs.

RTOS: μ C/FS does not assume the presence of a RTOS kernel. However, if you are using a RTOS, a simple port layer is required (consisting of a few semaphores), in order to prevent simultaneous access to core structures from different tasks. If you are not using a RTOS, this port layer may consist of empty functions.

1-2 TYPICAL USAGES

Applications have sundry reasons for non-volatile storage. A subset require (or benefit from) organizing data into named files within a directory hierarchy on a volume—basically, from having a file system. Perhaps the most obvious expose the structure of information to the user, like products that store images, video or music that are transferred to or from a PC. A web interface poses a similar opportunity, since the URLs of pages and images fetched by the remote browser would resolve neatly to locations on a volume.

Another typical use is data logging. A primary purpose of a device may be to collect data from its environment for later retrieval. If the information must persist across device reset events or will exceed the capacity of its RAM, some non-volatile memory is necessary. The benefit of a file system is the ability to organize that information logically, with a fitting directory structure, through a familiar API.

A file system can also store programs. In a simple embedded CPU, the program is stored at a fixed location in a non-volatile memory (usually flash). If an application must support firmware updates, a file system may be a more convenient place, since the software handles the details of storing the program. The boot-loader, of course, would need to be able to load the application, but since that requires only read-only access, no imposing program is required. The ROM boot-loaders in some CPUs can check the root directory of a SD card for a binary in addition to the more usual locations such as external NAND or NOR flash.

1-3 WHY FAT?

File Allocation Table (FAT) is a simple file system, widely supported across major OSs. While it has been supplanted as the format of hard drives in Windows PCs, removable media still use FAT because of its wide support. That is suitable for embedded systems, which would often be challenged to muster the resources for the modern file systems developed principally for large fixed disks.

μC/FS supports FAT because of the interoperability requirements of removable media, allowing that a storage medium be removed from an embedded device and connected to a PC. All variants and extensions are supported to specification.

A notorious weakness of FAT (exacerbated by early Windows system drivers) is its non-fail safe architecture. Certain operations leave the file system in an inconsistent state, albeit briefly, which may corrupt the disk or force a disk check upon unexpected power failure. μ C/FS minimizes the problem by ordering modifications wisely. The problem is completely solved in an optional journaling module which logs information about pending changes so those can be resumed on start-up after a power failure.

1-4 CHAPTER CONTENTS

Figure 1-1 shows the layout and flow of the book. This diagram should be useful to understand the relationship between chapters. The first (leftmost) column lists chapters that should be read in order to understand μ C/FS's structure. The chapters in the second column give greater detail about the application of μ C/FS. Each of the chapters in the third column examines a storage technology and its device driver. Finally, the fourth column lists the appendices, the topmost being the μ C/FS reference, configuration and porting manuals. Reference these sections regularly when designing a product using μ C/FS.

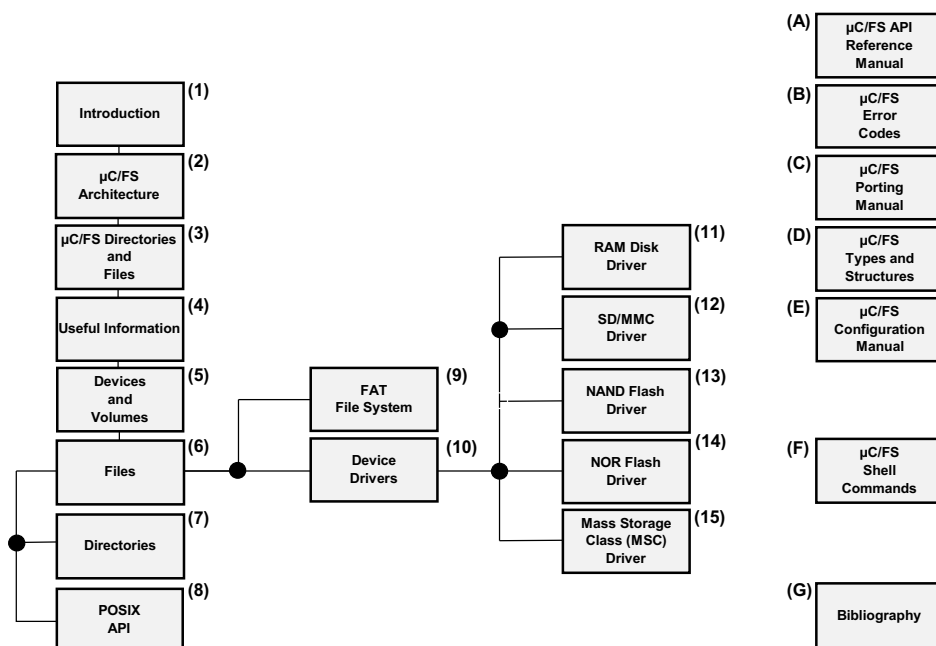


Figure 1-1 μ C/FS book layout

Chapter 1, “Introduction”. This chapter.

Chapter 2, “ μ C/FS Architecture”. This chapter contains a simplified block diagram of the various different μ C/FS modules and their relationships. The relationships are then explained.

Chapter 3, “ μ C/FS Directories and Files”. This chapter explains the directory structure and files needed to build a μ C/FS-based application. Learn about the files that are needed, where they should be placed, which module does what, and more.

Chapter 4, “Useful Information”. In this chapter, you will learn the nomenclature used in μ C/FS to access files and folders and the resources needed to use μ C/FS in your application.

Chapter 5, “Devices and Volumes”. Every file and directory accessed with μ C/FS is a constituent of a volume (a collection of files and directories) on a device (a physical or logical sector-addressed entity). This chapter explains how devices and volumes are managed.

Chapter 6, “Files”. μ C/FS complements the POSIX API with its own file access API. This chapter explains this API.

Chapter 7, “Directories”. μ C/FS complements the POSIX API with its own directory access API. This chapter explains this API.

Chapter 8, “POSIX API”. The best-known API for accessing and managing files and directories is specified within the POSIX standard (IEEE Std 1003.1), which is based in part in the ISO C standard (ISO/IEC 9899). This chapter explains how to use this API and examines some of its pitfalls and shortcomings.

Chapter 10, “FAT File System”. This chapter details the low-level architecture of the FAT file system. Though the API of μ C/FS is file system agnostic, the file system type does affect performance, reliability and security, as explained here as well.

Chapter 9, “Device Drivers”. All hardware accesses are eventually performed by a device driver. This chapter describes the drivers available with μ C/FS and broadly profiles supported media types in terms of cost, performance and complexity.

Chapter 11, “RAM Disk Driver”. This chapter demonstrates the use of the simplest storage medium, the RAM disk.

Chapter 12, “SD/MMC Drivers”. SD and MMC cards are flash-based removable storage devices commonly used in consumer electronics. For embedded CPUs, a SD/MMC card is an appealing medium because of its simple and widely-supported physical interfaces (one choice is SPI). This chapter describes the interface and function of these devices.

Chapter 13, “NAND Flash Driver”. NAND flash is the first category of flash media. Write speeds are fast (compared to NOR flash), at the expense of slower read speeds and complexities such as bit-errors and page program limitations. This chapter describes the functions of these devices and the architecture of the supporting driver.

Chapter 14, “NOR Flash Driver”. NOR flash is the second category of flash media. They suffer slow write speeds, balanced with blazingly-fast read speeds. Importantly, they are not plagued by the complications of NAND flash, which simplifies interfacing with them. This chapter describes the function of these devices and the architecture of the supporting driver.

Chapter 15, “MSC Driver”. The now-common USB drive implements the Mass Storage Class (MSC) protocol, and a CPU with a USB host interface can access these devices with appropriate software. The MSC driver, discussed in this chapter, with μ C/USB-Host is just such appropriate software.

Appendix A, “ μ C/FS API Reference”. The reference manual describes every API function. The arguments and return value of each function are given, supplemented by notes about its use and an example code listing.

Appendix B, “ μ C/FS Error Codes”. This appendix provides a brief explanation of μ C/FS error codes defined in `fs_err.h`.

Appendix C, “ μ C/FS Porting Manual”. The portability of μ C/FS relies upon ports to interface between its modules and the platform or environment. Most of the ports constitute the board support package (BSP), which is interposed between the file system suite (or driver) and hardware. The OS port adapts the software to a particular OS kernel. The porting manual describes each port function.

Appendix D, “ μ C/FS Types and Structures”. This appendix provides a reference to the μ C/FS types and structures.

Appendix E, “ μ C/FS Configuration”. μ C/FS is configured via defines in a single configuration file, `fs_cfg.h`. The configuration manual specifies each define and the meaning of possible values.

Appendix F, “Shell Commands”. A familiar method of accessing a file system, at least to engineers and computer scientists, is the command line. In an embedded system, a UART is a port over which commands can be executed easily, even for debug purposes. A set of shell commands have been developed for μ C/FS that mirror the syntax of UNIX utilities, as described in this chapter.

Appendix G, “Bibliography”.

Appendix H, “ μ C/FS Licensing Policy”.

µC/FS Architecture

µC/FS was written from the ground up to be modular and easy to adapt to different CPUs (Central Processing Units), RTOSs (Real-Time Operating Systems), storage media and compilers. Figure 2-1 shows a simplified block diagram of the different µC/FS modules and their relationships.

Notice that all of the µC/FS files start with '**fs_**'. This convention allows you to quickly identify which files belong to µC/FS. Also note that all functions and global variables start with '**FS**', and all macros and **#defines** start with '**FS_**'.

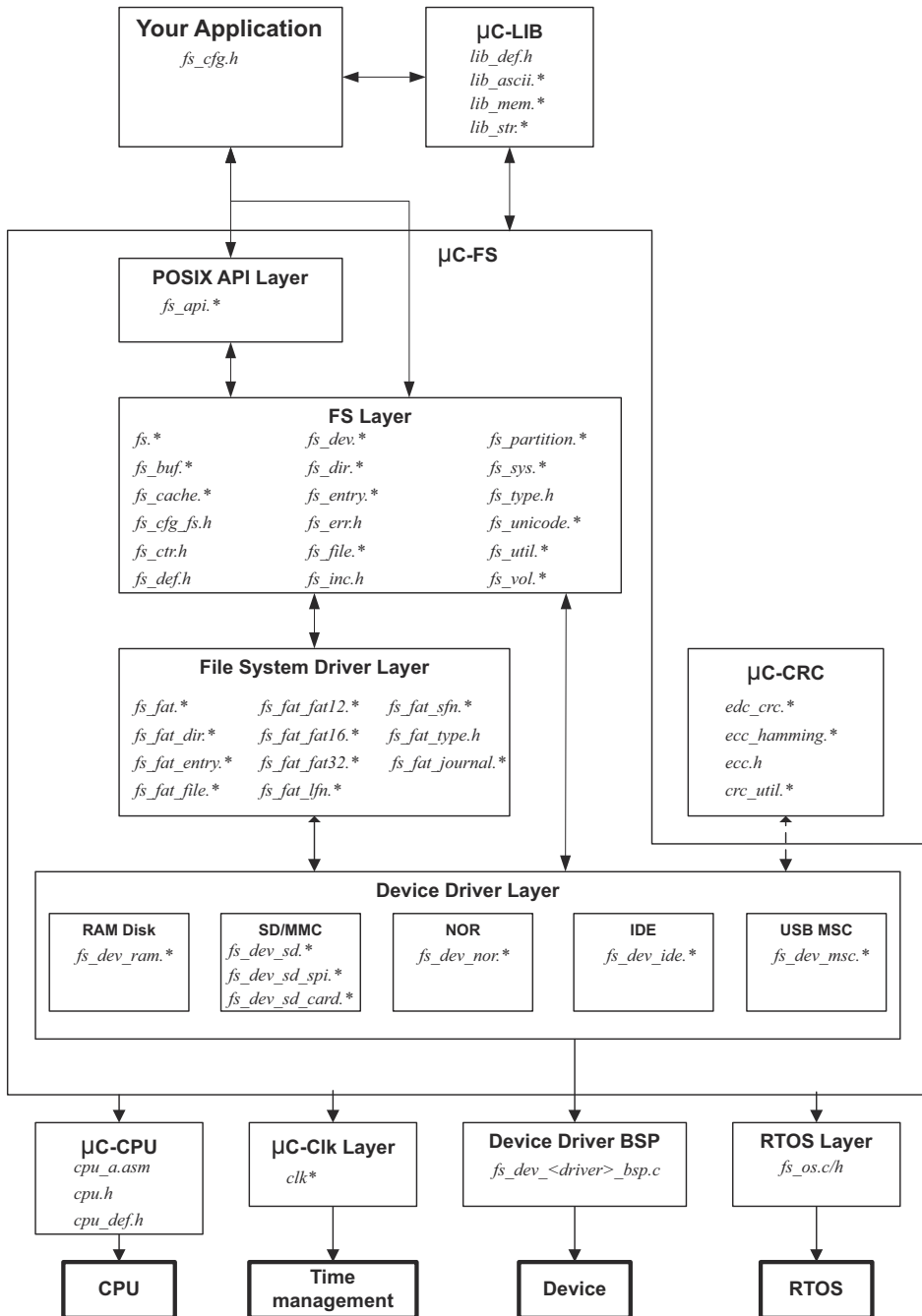


Figure 2-1 μC/FS architecture

2-1 ARCHITECTURE COMPONENTS

μC/Fs consists of a set of modular software components. It also requires a few external components (provided with the release) be compiled into the application and a few configuration and BSP files be adapted to the application.

2-1-1 YOUR APPLICATION

Your application needs to provide configuration information to μC/Fs in the form of one C header file named `fs_cfg.h`.

Some of the configuration data in `fs_cfg.h` consist of specifying whether certain features will be present. For example, LFN support, volume cache and file buffering are all enabled or disabled in this file. In all, there are about 30 `#define` to set. However, most of these can be set to their default values.

2-1-2 μC-LIB (LIBRARIES)

Because μC/Fs is designed to be used in safety critical applications, all 'standard' library functions like `strcpy()`, `memset()`, etc., have been re-written to follow the same quality as the rest of the file system software.

2-1-3 POSIX API LAYER

Your application interfaces to μC/Fs using the well-known `stdio.h` API (Application Programming Interface). Alternately, you can use μC/Fs's own file and directory interface functions. Basically, POSIX API layer is a layer of software that converts POSIX file access calls to μC/Fs file access calls.