A76xx Series Open SDK_文件系统_应用指导

LTE 模组

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
技术支持邮箱：support@simcom.com
官网：www.simcom.com

名称：	A76xx Series Open SDK_文件系统_应用指导
版本：	V1.01
类别：	应用文档
状态：	已发布

版权声明

芯讯通无线科技(上海)有限公司
上海市长宁区临虹路289号3号楼芯讯通总部大楼
电话：86-21-31575100
邮箱：simcom@simcom.com
官网：www.simcom.com

了解更多资料，请点击以下链接：
http://cn.simcom.com/download/list-230-cn.html

技术支持，请点击以下链接：
http://cn.simcom.com/ask/index-cn.html 或发送邮件至 support@simcom.com

Version History

Version	Date	Owner	What is new
V1.01	2023-06-27		第二版

About this Document

本文档适用于 A1803S open 系列、A1603 open 系列、A1606 open 系列。

目录

缩略语

FS	FileSystem
LittleFS	little Flash File System
FATFS	FAT File System
ANSI	American National Standard Institite
Unicode	Universal Multiple-Octet Coded Character Set

1FS 简介

1.1 FS 协议栈

A76xx 系列文件系统采用 LittleFS 和 FATFS 作为协议栈，其中 LittleFS 是面向 IoT 开发的嵌入式文件系统，专门为物联网中的模块而设计；而 FATFS 则是一个通用的文件系统(FAT/exFAT)模块，用于在小型嵌入式系统中实现支持多种存储媒介,有独立的缓冲区，可裁剪的文件系统。其中 Flash 模块是使用的 LittleFS 作为协议栈；而 SD 卡模块使用的是 FATFS 作为协议栈。

1.2 FS 特性

1）擦写均衡，littlefs 使用了 crc 作为随机数，实现随机擦写。
2）掉电保护，littlefs 结合日志和 COW 方式，提供了一套掉电保护策略。
3）低资源消耗，littlefs 占用资源比一般的小型嵌入式文件系统小。
4）FATFS 支持 ANSI / OEM 或 Unicode 中的长文件名。
5）FATFS 支持 SD 卡读写，可储存大量数据。

1.3 FS 目录结构

A76xx 系列的文件系统为方便用户使用，我们特意将本地存储空间映射为“C”盘符，用户可见的目录“C：/”为用户可操作目录，考虑到文件系统的特性，在“C”盘符下只支持创建和操作各类文件，不支持对目录进行操作，为此用户可使用 SD 卡（默认映射为“D”盘符）作为第二存储路径，支持用户对各种文件、目录进行操作，同时也是方便用户进行大量数据的读写操作。

1.4 FS 功能支持

A76xx 系列的文件系统提供了以下各功能：
文件操作：open，read，write，close，sync，seek，tell，traverse，delete 和 size 等。
目录操作：mkdir，rmdir，open，read，close，traverse，pathexist 等

其他功能：split，rename，save，getdisksize 等

2 FS API 介绍

鉴于 A76xx 系列文件系统采用了不同的嵌入式文件系统，为了方便用户使用，因此提供了和标准 C 库类似的 API 封装接口，而摒弃了 LittleFS 和 FATFS 的原生接口，用法上尽量向标准 C 库看齐。

头文件：#include “PAL_file.h”

2.1 sAPI_fformat

此接口用于格式化 SD 卡或外挂 flash，仅支持“D:”。

接口:	int **sAPI_fformat**(char *vol);
输入:	**vol：** 需要格式化的磁盘
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.2 sAPI_fopen

此接口用于打开或创建文件并将其与流关联，支持"C:"、"D:"。

接口:	SCFILE ***sAPI_fopen**(const char *fname, const char *mode);
输入:	**fname：** 包含路径的完整文件名 **mode：** 类型规定的 const 字符串，r/w/a/b **rb：** 只读 **wb / ab：** 只写 **rb+ / wb+ /ab +：** 读写
输出:	无
返回值:	文件流的指针。失败时返回 NULL。

NOTE:	mode 通常使用“rb”和“wb”模式；

2.3 sAPI_fclose

此接口用于关闭文件流，支持"C:"、"D:"。

接口:	int **sAPI_fclose**(SCFILE *fp);
输入:	**fp**：要关闭的文件的流索引
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.4 sAPI_fwrite

此接口用于将数据写入文件，支持“C:”、“D:“。

接口:	int **sAPI_fwrite**(const void *buffer, size_t size, size_t num, SCFILE *fp);
输入:	**buffer**：要写入文件的数据的指针 **size**：要写入的每个数据的大小，以字节为单位 **num**：数据的数量 **fp**：要写入文件的流指针
输出:	无
返回值:	实际写入的数据长度，是一个整数
NOTE:	无

2.5 sAPI_fread

此接口用于读取特定长度的数据到缓存区，支持“C:”、“D:”。

接口:	int **sAPI_fread**(void *buffer, size_t size, size_t num, SCFILE *fp);
输入:	**buffer**：指向缓冲区的指针，用来存放需要写入的数据 **size**：要读取的每个数据的大小，以字节为单位 **num**：数据的数量 **fp**：要读取文件的流指针
输出:	无
返回值:	成功读取的数据总数，是一个整数
NOTE:	无

2.6 sAPI_fseek

此接口用于设置流文件指定的文件位置指示器。通过向参数：whence 处添加起始位置，通过 offset 可以获得从 whence 开始的以字节为单位的新位置，支持“C:”、“D:“。

接口:	int **sAPI_fseek**(SCFILE *fp, long offset, int whence);
输入:	**fp**：文件流索引 **offset**：从起始地址移动大小 **whence**：参数可以是： FS_SEEK_BEGIN 0 FS_SEEK_CURREN 1 FS_SEEK_END 2
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.7 sAPI_ftell

此接口用于获取文件位置，支持“C:”、“D:”。

接口:	long **sAPI_ftell**(SCFILE *fp);

输入:	**fp**：文件流索引
输出:	无
返回值:	当前文件位置
NOTE:	无

2.8 sAPI_frewind

此接口用于将文件指针移动到文件的开头，支持"C:"、"D:"。

接口:	int **sAPI_frewind**(SCFILE *fp);
输入:	**fp**：文件流索引
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.9 sAPI_fsize

此接口用于获取打开文件的大小，支持"C:"、"D:"。

接口:	int **sAPI_fsize**(SCFILE *fp);
输入:	**fp**：文件流索引
输出:	无
返回值:	文件的大小
NOTE:	无

2.10 sAPI_fsync

此接口用于同步文件，支持“C:”、“D:”。

接口:	int **sAPI_fsync**(SCFILE *fp);
输入:	**fp**：文件流索引
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.11 sAPI_mkdir

此接口用于创建一个新目录，支持“C:”、“D:”。

接口:	int **sAPI_mkdir**(const char *path, unsigned int mode);
输入:	**path**：带路径的文件夹名 **mode**：创建模式（没有实际意义，设置为 0）
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.12 sAPI_opendir

此接口用于打开一个目录，支持“C:”、“D:”。

接口:	SCDIR ***sAPI_opendir**(const char *path);
输入:	**path**：带路径的目录名
输出:	无
返回值:	目录流的指针。失败时返回 NULL。
NOTE:	无

2.13 sAPI_closedir

此接口用于关闭一个目录，支持“C:”、“D:”。

接口:	int **sAPI_closedir**(SCDIR *dirp);
输入:	**dirp**：目录流
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.14 sAPI_readdir

此接口用于读取一个目录，支持“C:”、“D:”。

接口:	struct dirent ***sAPI_readdir**(SCDIR *dirp);
输入:	**drip**：目录流
输出:	无
返回值:	指向包含文件信息（文件名、类型、大小）的结构的指针。失败时返回 NULL。
NOTE:	无

2.15 sAPI_seekdir

此接口用于为下一次调 sAPI_readdir()设置读取位置，仅支持“C:”。

接口:	int **sAPI_seekdir**(SCDIR *dirp, unsigned long offset);
输入:	**drip**：目录流 **offset**：从目录开始的移动大小
输出:	无

返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.16 sAPI_telldir

此接口用于获取目录流的当前位置，仅支持“C:”。

接口:	int **sAPI_telldir**(SCDIR *dirp);
输入:	**drip**：目录流
输出:	无
返回值:	当前位置
NOTE:	无

2.17 sAPI_remove

此接口用于删除一个文件或目录，支持“C:”、“D:”。

接口:	int **sAPI_remove**(const char *fname);
输入:	**fname**：文件或目录名
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.18 sAPI_rename

此接口用于重命名一个文件，支持“C:”、“D:”。

接口:	int **sAPI_rename**(const char *oldpath, const char *newpath);
输入:	**oldpath**：旧文件名 **newpath**：新文件名
输出:	无
返回值:	0：过程成功执行 other：过程执行失败
NOTE:	无

2.19 sAPI_access

此接口用于判断文件是否存在，支持“C:”、“D:”。

接口:	int **sAPI_access**(const char *path, int mode);
输入:	**path**：文件名 **mode**：模式（没有实际意义，设置为 0）
输出:	无
返回值:	0：文件存在 other：文件不存在
NOTE:	无

2.20 sAPI_stat

此接口用于获取现有文件或目录的信息结构，支持“C:”、“D:”。

接口:	int **sAPI_stat**(const char *path, struct dirent *info);
输入:	**path**：文件或目录名 **info**：包含文件信息（文件名、类型、文件大小）的结构
输出:	无
返回值:	0：文件存在

	other：文件不存在
NOTE:	无

2.21 sAPI_GetSize

此接口用于获取文件系统的空间大小，支持“C:”、“D:”。

接口:	long long **sAPI_GetSize**(char *disc);
输入:	**disc**：磁盘名（例如“C”）
输出:	无
返回值:	磁盘总大小
NOTE:	无

2.22 sAPI_GetFreeSize

此接口用于获取文件系统的可用空间大小，支持“C:”、“D:”。

接口:	long long **sAPI_GetFreeSize**(char *disc);
输入:	**disc**：磁盘名
输出:	无
返回值:	磁盘空闲大小
NOTE:	无

2.23 sAPI_GetUsedSize

此接口用于获取文件系统已使用的空间大小，支持“C:”、“D:”。

接口:	long long **sAPI_GetUsedSize**(char *disc);
输入:	**disc**：磁盘名
输出:	无
返回值:	磁盘使用大小
NOTE:	无

3 数据定义

3.1 Struct Dirent

```
typedef struct dirent
{
    unsigned char type;
    unsigned long size;
    char name[MAX_FULL_NAME_LENGTH];
}dirent_t;
```

3.2 SCFILE

```
typedef struct
{
    void *disk;
    lfs_file_t file;
    SC_FILE_LOCATION loc;
    unsigned int fatfs_file_handle;
} SCFILE;
```

3.3 SCDIR

```
typedef struct
{
    void *disk;
    lfs_dir_t dir;
    struct lfs_info info;
    SC_FILE_LOCATION loc;
#ifdef FEATURE_SIMCOM_SD_CARD
```

```
    unsigned int handle;
    char path[MAX_FULL_NAME_LENGTH];
    int fileIndex;
#endif
} SCDIR;
```

3.4 SC_FILE_LOCATION

```
typedef enum{
    LOC_FLASH,
    LOC_EXT_FLASH,
    LOC_SD_CARD,
} SC_FILE_LOCATION;
```

4FS Demo

```
#include "simcom_file_system.h"
#define   BUFF_LEN 100

void FSDemo(void)
{
    SCFILE *file_hdl = NULL;
    SCDIR *dir_hdl = NULL;
    UINT32 ret = 0;

    unsigned long buff_data_len = 0;
    int actul_write_len = 0;
    int actul_read_len = 0;
    INT64 total_size = 0;
    INT64 free_size = 0;
    INT64 used_size = 0;
    struct dirent_t *info_dir = NULL;
    struct dirent_t info_file = {0};

    char *file_name = "D:/test_file.txt";
    UINT32 opt = 0;
    char *note = "\r\nPlease select an option to test from the items listed below.\r\n";
    char *options_list[] =
    {
        "1. Write file",
        "2. Read file",
        "3. Delete file",
        "4. Rename file",
        "5. Get disk size",
```

```
        "6. Make directory",
        "7. Remove directory",
        "8. List file",
        "9. Format drive D:",
        "99. Back",
};

while (1)
{
    ui_print("%s", note);
    PrintfOptionMenu(options_list, sizeof(options_list) / sizeof(options_list[0]));

    opt = UartReadValue();
    sAPI_Debug("opt %d", opt);

    switch (opt)
    {
        case SC_FS_DEMO_WRITEFILE:
        {
            char write_str[1000] = {0};
            char fileName[100];
            ui_print("\r\nPlease input fileName\r\n");
            UartReadLine(fileName, 100);
            ui_print("\r\nPlease input data\r\n");
            UartReadLine(write_str, 1000);
            file_hdl = sAPI_fopen(fileName, "wb+");
            if (file_hdl == NULL)
            {
                ui_print("open %s fail", fileName);
                break;
            }
            buff_data_len = strlen(write_str);
            actul_write_len = sAPI_fwrite(write_str, 1, buff_data_len, file_hdl);
```

```
        if (actul_write_len != buff_data_len)
        {
            sAPI_fclose(file_hdl);
            ui_print("%s write fail [%d]", fileName, actul_write_len);
            break;
        }
        ret = sAPI_fclose(file_hdl);
        if (ret != 0)
        {
            ui_print("%s close fail [%d]", fileName, ret);
            break;
        }
        else
        {
            file_hdl = NULL;
        }
        break;
    }
    case SC_FS_DEMO_READFILE:
    {
        int position;
        char fileName[100];
        char read_buffer[MAX_OUTPUT_LEN];
        ui_print("\r\nPlease input fileName\r\n");
        UartReadLine(fileName, 100);
        file_hdl = sAPI_fopen(fileName, "rb");
        if (file_hdl == NULL)
        {
            ui_print("open %s fail", fileName);
            break;
        }

        buff_data_len = sAPI_fsize(file_hdl);
```

```
                ui_print("sAPI_fsize buff_data_len: %d",    buff_data_len);


                buff_data_len = (buff_data_len <= MAX_OUTPUT_LEN) ? buff_data_len :
MAX_OUTPUT_LEN;
                memset(read_buffer, 0, MAX_OUTPUT_LEN);
                actul_read_len = sAPI_fread(read_buffer, 1, buff_data_len, file_hdl);
                if (actul_read_len <= 0)
                {
                    sAPI_fclose(file_hdl);
                    ui_print("%s read fail [%d]", fileName, actul_read_len);
                    break;
                }
                ui_print("read len: %d,data: %s", actul_read_len, read_buffer);
                sAPI_fseek(file_hdl, 1, 0);
                position = sAPI_ftell(file_hdl);
                ui_print("postion: %d", position);
                memset(read_buffer, 0, MAX_OUTPUT_LEN);
                actul_read_len = sAPI_fread(read_buffer, 1, buff_data_len, file_hdl);
                ui_print("read len: %d,data: %s", actul_read_len, read_buffer);
                ret = sAPI_fclose(file_hdl);
                if (ret != 0)
                {
                    ui_print("close %s fail", file_name);
                    break;
                }
                else
                {
                    file_hdl = NULL;
                }
                break;
            }
            case SC_FS_DEMO_DELETEFILE:
            {
```

```
        char fileName[100];
        ui_print("\r\nPlease input fileName\r\n");
        UartReadLine(fileName, 100);
        ret = sAPI_stat(fileName, &info_file);
        if (ret != 0)
        {
            ui_print("sAPI_stat fail [%d]", ret);
            break;
        }
        ui_print("[%s]-size[%d]-type[%d]", info_file.name, info_file.size, info_file.type);
        ret = sAPI_remove(fileName);
        if (ret != 0)
        {
            ui_print("remove %s fail [%d]", fileName, ret);
            break;
        }
        ui_print("remove %s sucess", fileName);
        ret = sAPI_stat(fileName, &info_file);
        if (ret != 0)
        {
            ui_print("not find file");
            break;
        }
        ui_print("[%s]-size[%d]-type[%d]", info_file.name, info_file.size, info_file.type);

        break;
    }
    case SC_FS_DEMO_RENAMEFILE:
    {
        char fileName[100];
        char new_name[100];
        ui_print("\r\nPlease input fileName\r\n");
        UartReadLine(fileName, 100);
```

```
ui_print("\r\nPlease input new fileName\r\n");
UartReadLine(new_name, 100);
ret = sAPI_stat(fileName, &info_file);
if (ret != 0)
{
    ui_print("sAPI_stat fail [%d]", ret);
    break;
}
ui_print("[%s]-size[%d]-type[%d]", info_file.name, info_file.size, info_file.type);
ret = sAPI_stat(new_name, &info_file);
if (ret != 0)
{
    ui_print("not find file");
}
else
{
    ui_print("[%s]-size[%d]-type[%d]", info_file.name, info_file.size, info_file.type);
    break;
}
ret = sAPI_rename(fileName, new_name);
if (ret != 0)
{
    ui_print("rename %s fail [%d]", fileName, ret);
    break;
}
ui_print("rename %s sucess", new_name);
ret = sAPI_stat(fileName, &info_file);
if (ret != 0)
{
    ui_print("not find file");
}
ret = sAPI_stat(new_name, &info_file);
if (ret != 0)
```

```
            {
                ui_print("sAPI_stat fail [%d]", ret);
                break;
            }
            ui_print("[%s]-size[%d]-type[%d]", info_file.name, info_file.size, info_file.type);

            break;
        }
        case SC_FS_DEMO_GETDISKSIZE:
        {
            int disk = 0;
            ui_print("\r\nPlease input disk:0 is C:/ and 1 is D:/\r\n");
            disk = UartReadValue();
            if (disk)
            {
                total_size = sAPI_GetSize("D:/");
                free_size = sAPI_GetFreeSize("D:/");
                used_size = sAPI_GetUsedSize("D:/");
            }
            else
            {
                total_size = sAPI_GetSize("C:/");
                free_size = sAPI_GetFreeSize("C:/");
                used_size = sAPI_GetUsedSize("C:/");
            }
            ui_print("total_size= %lld, free_size= %lld,used_size = %lld", total_size, free_size,
used_size);

            break;
        }
        case SC_FS_DEMO_MAKEDIR:
        {
            char dir_name[100];
```

```
        ui_print("\r\nPlease input dir\r\n");
        UartReadLine(dir_name, 100);
        ret = sAPI_mkdir(dir_name, 0);
        if (ret != 0)
        {
            ui_print("creat %s fail [%d]", dir_name, ret);
            break;
        }
        ui_print("creat %s sucess", dir_name);
        ret = sAPI_stat(dir_name, &info_file);
        if (ret != 0)
        {
            ui_print("get %s info fail[%d]", dir_name, ret);
            break;
        }
        ui_print("[%s]-[%ld]-[%d]", info_file.name, info_file.size, info_file.type);
        break;
    }
    case SC_FS_DEMO_REMOVEDIR:
    {
        char dir_name[100];
        ui_print("\r\nPlease input dir\r\n");
        UartReadLine(dir_name, 100);
        ret = sAPI_remove(dir_name);
        if (ret != 0)
        {
            ui_print("delte %s fail [%d]", dir_name, ret);
            break;
        }
        ui_print("delte %s success", dir_name);
        break;
    }
    case SC_FS_DEMO_OPENDIR:
```

```
        {
            char dir_name[100];
            ui_print("\r\nPlease input dir\r\n");
            UartReadLine(dir_name, 100);
            dir_hdl = sAPI_opendir(dir_name);
            if (dir_hdl == NULL)
            {
                ui_print("open %s fail", dir_name);
                break;
            }
            ui_print("open %s success", dir_name);

            while ((info_dir = sAPI_readdir(dir_hdl)) != NULL)
            {
                ui_print("name[%s]-size[%ld]-type[%d]\r\n", info_dir->name, info_dir->size,
info_dir->type);
            }

            ret = sAPI_closedir(dir_hdl);
            if (ret != 0)
            {
                ui_print("close %s fail[%d]", dir_name, ret);
                break;
            }
            break;
        }
        case SC_FS_DEMO_FORANT_DIR:
        {
            char drive[10] = {0};
            ui_print("\r\nPlease input format drive\r\n");
            UartReadLine(drive, 10);
            ui_print("drive: [%s] ", drive);
            ret = sAPI_fformat(drive);
```

```
    if (0 == ret)
    {
        ui_print("\r\nFormat success!\r\n");
    }
    else
    {
        ui_print("\r\nFormat fail!\r\n");
        break;
    }

    char dir_name[100];
    ui_print("\r\nPlease input dir\r\n");
    UartReadLine(dir_name, 100);
    dir_hdl = sAPI_opendir(dir_name);
    if (dir_hdl == NULL)
    {
        ui_print("open %s fail", dir_name);
        break;
    }
    ui_print("open %s success", dir_name);

    while ((info_dir = sAPI_readdir(dir_hdl)) != NULL)
    {
        ui_print("name[%s]-size[%ld]-type[%d]\r\n", info_dir->name, info_dir->size,
info_dir->type);
    }

    ret = sAPI_closedir(dir_hdl);
    if (ret != 0)
    {
        ui_print("close %s fail[%d]", dir_name, ret);
        break;
    }
```

```
                break;
            }
            case 99:
            {
                return;
            }
            default:
            {
                break;
            }
        }
    }
}
```