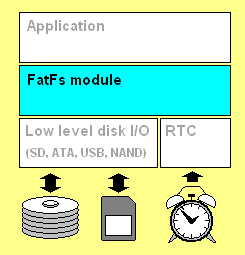
**FatFs**

FatFS是一个为小型嵌入式系统设计的通用FAT(File Allocation Table)文件系统模块。FatFs 的编写遵循ANSI C，并且完全与磁盘I/O层分开。因此，它独立(不依赖)于硬件架构。它可以被嵌入到低成本的微控制器中，如AVR, 8051, PIC, ARM, Z80, 68K 等等，而不需要做任何修改。



**特点**

* Windows兼容的FAT文件系统
* 不依赖于平台，易于移植
* 代码和工作区占用空间非常小
* 多种配置选项：

多卷(物理驱动器和分区)

多ANSI/OEM代码页，包括DBCS

在ANSI/OEM或Unicode中长文件名的支持

RTOS的支持

多扇区大小的支持

只读，最少API，I/O缓冲区等等

**应用程序接口**

FatFs 模块为应用程序提供了下列函数，这些函数描述了FatFs能对FAT卷执行哪些操作。

**f\_mount**

在FatFs模块上注册/注销一个工作区(文件系统对象)

FRESULT f\_mount (

BYTE Drive, /\* 逻辑驱动器号 \*/

FATFS\* FileSystemObject /\* 工作区指针 \*/

);

参数

Drive注册/注销工作区的逻辑驱动器号(0-9)。

FileSystemObject工作区(文件系统对象)指针。

返回值

FR\_OK (0)函数成功。

FR\_INVALID\_DRIVE驱动器号无效

描述

f\_mount函数在FatFs模块上注册/注销一个工作区。 在使用任何其他文件函数之前，必须使用该函数为每个卷注册一个工作区。要注销一个工作区，只要指定FileSystemObject为NULL即可，然后该工作区可以被丢弃。

该函数只初始化给定的工作区，以及将该工作区的地址注册到内部表中，不访问磁盘I/O层。卷装入过程是在f\_mount函数后或存储介质改变后的第一次文件访问时完成的。

**f\_open**

创建/打开一个用于访问文件的文件对象

FRESULT f\_open (

FIL\* FileObject, /\* 空白文件对象结构指针 \*/

const XCHAR\* FileName, /\* 文件名指针 \*/

BYTE ModeFlags /\* 模式标志 \*/

);

参数

FileObject将被创建的文件对象结构的指针。

FileName

NULL结尾的字符串指针，该字符串指定了将被创建或打开的文件名。

ModeFlags指定文件的访问类型和打开方法。它是由下列标志的一个组合指定的。

|  |  |
| --- | --- |
| **模式** | **描述** |
| **FA\_READ** | 指定读访问对象。可以从文件中读取数据。 与FA\_WRITE结合可以进行读写访问。 |
| **FA\_WRITE** | 指定写访问对象。可以向文件中写入数据。 与FA\_READ结合可以进行读写访问。 |
| **FA\_OPEN\_EXISTING** | 打开文件。如果文件不存在，则打开失败。(默认) |
| **FA\_OPEN\_ALWAYS** | 如果文件存在，则打开；否则，创建一个新文件。 |
| **FA\_CREATE\_NEW** | 创建一个新文件。如果文件已存在，则创建失败。 |
| **FA\_CREATE\_ALWAYS** | 创建一个新文件。如果文件已存在，则它将被截断并覆盖。 |

注意：当 \_FS\_READONLY == 1 时，模式标志 FA\_WRITE, FA\_CREATE\_ALWAYS, FA\_CREATE\_NEW, FA\_OPEN\_ALWAYS 是无效的。

返回值

FR\_OK (0)函数成功，该文件对象有效。

FR\_NO\_FILE找不到该文件。

FR\_NO\_PATH找不到该路径。

FR\_INVALID\_NAME文件名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_EXIST该文件已存在。

FR\_DENIED由于下列原因，所需的访问被拒绝：

* 以写模式打开一个只读文件。
* 由于存在一个同名的只读文件或目录，而导致文件无法被创建。
* 由于目录表或磁盘已满，而导致文件无法被创建。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_WRITE\_PROTECTED在存储介质被写保护的情况下，以写模式打开或创建文件对象。

FR\_DISK\_ERR由于底层磁盘I/O接口函数中的一个错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效地FAT卷。

描述

如果函数成功，则创建一个文件对象。该文件对象被后续的读/写函数用来访问文件。如果想要关闭一个打开的文件对象，则使用f\_close函数。如果不关闭修改后的文件，那么文件可能会崩溃。

在使用任何文件函数之前，必须使用f\_mount函数为驱动器注册一个工作区。只有这样，其他文件函数才能正常工作。

例子(文件拷贝)

void main (void)

{

FATFS fs[2]; /\* 逻辑驱动器的工作区(文件系统对象) \*/

FIL fsrc, fdst; /\* 文件对象 \*/

BYTE buffer[4096]; /\* 文件拷贝缓冲区 \*/

FRESULT res; /\* FatFs 函数公共结果代码 \*/

UINT br, bw; /\* 文件读/写字节计数 \*/

/\* 为逻辑驱动器注册工作区 \*/

f\_mount(0, &fs[0]);

f\_mount(1, &fs[1]);

/\* 打开驱动器 1 上的源文件 \*/

res = f\_open(&fsrc, "1:srcfile.dat", FA\_OPEN\_EXISTING | FA\_READ);

if (res) die(res);

/\* 在驱动器 0 上创建目标文件 \*/

res = f\_open(&fdst, "0:dstfile.dat", FA\_CREATE\_ALWAYS | FA\_WRITE);

if (res) die(res);

/\* 拷贝源文件到目标文件 \*/

for (;;) {

res = f\_read(&fsrc, buffer, sizeof(buffer), &br);

if (res || br == 0) break; /\* 文件结束错误 \*/

res = f\_write(&fdst, buffer, br, &bw);

if (res || bw < br) break; /\* 磁盘满错误 \*/

}

/\* 关闭打开的文件 \*/

f\_close(&fsrc);

f\_close(&fdst);

/\* 注销工作区(在废弃前) \*/

f\_mount(0, NULL);

f\_mount(1, NULL);

}

**f\_close**

关闭一个打开的文件

FRESULT f\_close (

FIL\* FileObject /\* 文件对象结构的指针 \*/

);

参数

FileObject指向将被关闭的已打开的文件对象结构的指针。

返回值

FR\_OK (0)文件对象已被成功关闭。>FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_INVALID\_OBJECT文件对象无效。

描述

f\_close函数关闭一个打开的文件对象。无论向文件写入任何数据，文件的缓存信息都将被写回到磁盘。该函数成功后，文件对象不再有效，并且可以被丢弃。如果文件对象是在只读模式下打开的，不需要使用该函数，也能被丢弃。

**f\_read** 从一个文件读取数据

FRESULT f\_read (

FIL\* FileObject, /\* 文件对象结构的指针 \*/

void\* Buffer, /\* 存储读取数据的缓冲区的指针 \*/

UINT ByteToRead, /\* 要读取的字节数 \*/

UINT\* ByteRead /\* 返回已读取字节数变量的指针 \*/

);

参数

FileObject指向将被读取的已打开的文件对象结构的指针。

Buffer指向存储读取数据的缓冲区的指针。

ByteToRead要读取的字节数，UINT范围内。

ByteRead指向返回已读取字节数的UINT变量的指针。在调用该函数后，无论结果如何，数值都是有效的。

返回值

FR\_OK (0)函数成功。

FR\_DENIED由于文件是以非读模式打开的，而导致该函数被拒绝。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_INVALID\_OBJECT文件对象无效。

描述

文件对象中的读/写指针以已读取字节数增加。该函数成功后，应该检查 \*ByteRead 来检测文件是否结束。在读操作过程中，一旦 \*ByteRead < ByteToRead ，则读/写指针到达了文件结束位置。

**f\_write** 写入数据到一个文件

FRESULT f\_write (

FIL\* FileObject, /\* 文件对象结构的指针 \*/

const void\* Buffer, /\* 存储写入数据的缓冲区的指针 \*/

UINT ByteToWrite, /\* 要写入的字节数 \*/

UINT\* ByteWritten /\* 返回已写入字节数变量的指针 \*/

);

参数

FileObject指向将被写入的已打开的文件对象结构的指针。

Buffer指向存储写入数据的缓冲区的指针。

ByteToRead要写入的字节数，UINT范围内。

ByteRead指向返回已写入字节数的UINT变量的指针。在调用该函数后，无论结果如何，数值都是有效的。

返回值

FR\_OK (0)函数成功。

FR\_DENIED由于文件是以非写模式打开的，而导致该函数被拒绝。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_INVALID\_OBJECT文件对象无效。

描述

文件对象中的读/写指针以已写入字节数增加。该函数成功后，应该检查 \*ByteWritten 来检测磁盘是否已满。在写操作过程中，一旦 \*ByteWritten < \*ByteToWritten ，则意味着该卷已满。

**f\_lseek**

移动一个打开的文件对象的文件读/写指针。也可以被用来扩展文件大小(簇预分配)。

FRESULT f\_lseek (

FIL\* FileObject, /\* 文件对象结构指针 \*/

DWORD Offset /\* 文件字节偏移 \*/

);

参数

FileObject打开的文件对象的指针

Offset相对于文件起始处的字节数

返回值

FR\_OK (0)函数成功。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_INVALID\_OBJECT文件对象无效。

描述

f\_lseek函数当FS\_MINIMIZE <= 2时可用。

offset只能被指定为相对于文件起始处的字节数。当在写模式下指定了一个超过文件大小的offset时，文件的大小将被扩展，并且该扩展的区域中的数据是未定义的。这适用于为快速写操作迅速地创建一个大的文件。f\_lseek函数成功后，为了确保读/写指针已被正确地移动，必须检查文件对象中的成员fptr。如果fptr不是所期望的值，则发生了下列情况之一。

* 文件结束。指定的offset被钳在文件大小，因为文件已被以只读模式打开。
* 磁盘满。卷上没有足够的空闲空间去扩展文件大小。

例子

/\* 移动文件读/写指针到相对于文件起始处偏移为5000字节处 \*/

res = f\_lseek(file, 5000);

/\* 移动文件读/写指针到文件结束处，以便添加数据 \*/

res = f\_lseek(file, file->fsize);

/\* 向前3000字节 \*/

res = f\_lseek(file, file->fptr + 3000);

/\* 向后(倒带)2000字节(注意溢出) \*/

res = f\_lseek(file, file->fptr - 2000);

/\* 簇预分配(为了防止在流写时缓冲区上溢 \*/

res = f\_open(file, recfile, FA\_CREATE\_NEW | FA\_WRITE); /\* 创建一个文件 \*/

res = f\_lseek(file, PRE\_SIZE); /\* 预分配簇 \*/

if (res || file->fptr != PRE\_SIZE) ... /\* 检查文件大小是否已被正确扩展 \*/

res = f\_lseek(file, DATA\_START); /\* 没有簇分配延迟地记录数据流 \*/

...

res = f\_truncate(file); /\* 截断未使用的区域 \*/

res = f\_lseek(file, 0); /\* 移动到文件起始处 \*/

...

res = f\_close(file);

**f\_truncate**

截断文件大小

FRESULT f\_truncate (

FIL\* FileObject /\* 文件对象结构指针 \*/

);

参数

FileObject待截断的打开的文件对象的指针。

返回值

FR\_OK (0函数成功。

FR\_DENIED由于文件是以非写模式打开的，而导致该函数被拒绝。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_INVALID\_OBJECT文件对象无效。

描述

f\_truncate函数当\_FS\_READONLY == 0 并且 \_FS\_MINIMIZE == 0时可用。

f\_truncate函数截断文件到当前的文件读/写指针。当文件读/写指针已经指向文件结束时，该函数不起作用。

**f\_sync**

冲洗一个写文件的缓存信息

FRESULT f\_sync (

FIL\* FileObject /\* 文件对象结构的指针 \*/

);

参数

FileObject待冲洗的打开的文件对象的指针。

返回值

FR\_OK (0)函数成功。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_INVALID\_OBJECT文件对象无效。

描述

f\_sync函数当\_FS\_READONLY == 0时可用。

f\_sync函数和f\_close函数执行同样的过程，但是文件仍处于打开状态，并且可以继续对文件执行读/写/移动指针操作。这适用于以写模式长时间打开文件，比如数据记录器。定期的或f\_write后立即执行f\_sync可以将由于突然断电或移去磁盘而导致数据丢失的风险最小化。在f\_close前立即执行f\_sync没有作用，因为在f\_close中执行了f\_sync。换句话说，这两个函数的差异就是文件对象是不是无效的。

**f\_opendir**

打开一个目录

FRESULT f\_opendir (

DIR\* DirObject, /\* 空白目录对象结构的指针 \*/

const XCHAR\* DirName /\* 目录名的指针 \*/

);

参数

DirObject待创建的空白目录对象的指针。

DirName'\0'结尾的字符串指针，该字符串指定了将被打开的目录名。

返回值

FR\_OK (0)函数成功，目录对象被创建。该目录对象被后续调用，用来读取目录项。

FR\_NO\_PATH找不到路径。

FR\_INVALID\_NAME路径名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_opendir函数当\_FS\_MINIMIZE <= 1时可用。

f\_opendir函数打开一个已存在的目录，并为后续的调用创建一个目录对象。该目录对象结构可以在任何时候不经任何步骤而被丢弃。

**f\_readdir**

读取目录项

FRESULT f\_readdir (

DIR\* DirObject, /\* 指向打开的目录对象结构的指针 \*/

FILINFO\* FileInfo /\* 指向文件信息结构的指针 \*/

);

参数

DirObject打开的目录对象的指针。

FileInfo存储已读取项的文件信息结构指针。

返回值

FR\_OK (0)函数成功。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_INVALID\_OBJECT文件对象无效。

描述

f\_readdir函数当\_FS\_MINIMIZE <= 1时可用。

f\_readdir函数顺序读取目录项。目录中的所有项可以通过重复调用f\_readdir函数被读取。当所有目录项已被读取并且没有项要读取时，该函数没有任何错误地返回一个空字符串到f\_name[]成员中。当FileInfo给定一个空指针时，目录对象的读索引将被回绕。

当LFN功能被使能时，在使用f\_readdir函数之前，文件信息结构中的lfname和lfsize必须被初始化为有效数值。lfname是一个返回长文件名的字符串缓冲区指针。lfsize是以字符为单位的字符串缓冲区的大小。如果读缓冲区或LFN工作缓冲区的大小(对于LFN)不足，或者对象没有LFN，则一个空字符串将被返回到LFN读缓冲区。如果LFN包含任何不能被转换为OEM代码的字符，则一个空字符串将被返回，但是这不是Unicode API配置的情况。当lfname是一个空字符串时，没有LFN的任何数据被返回。当对象没有LFN时，任何小型大写字母可以被包含在SFN中。

当相对路径功能被使能(\_FS\_RPATH == 1)时，"."和".."目录项不会被过滤掉，并且它将出现在读目录项中。

例子

FRESULT scan\_files (char\* path)

{

FRESULT res;

FILINFO fno;

DIR dir;

int i;

char \*fn;

#if \_USE\_LFN

static char lfn[\_MAX\_LFN \* (\_DF1S ? 2 : 1) + 1];

fno.lfname = lfn;

fno.lfsize = sizeof(lfn);

#endif

res = f\_opendir(&dir, path);

if (res == FR\_OK) {

i = strlen(path);

for (;;) {

res = f\_readdir(&dir, &fno);

if (res != FR\_OK || fno.fname[0] == 0) break;

if (fno.fname[0] == '.') continue;

#if \_USE\_LFN

fn = \*fno.lfname ? fno.lfname : fno.fname;

#else

fn = fno.fname;

#endif

if (fno.fattrib & AM\_DIR) {

sprintf(&path[i], "/%s", fn);

res = scan\_files(path);

if (res != FR\_OK) break;

path[i] = 0;

} else {

printf("%s/%s\n", path, fn);

}

}

}

return res;

}

**f\_getfree**

获取空闲簇的数目

FRESULT f\_getfree (

const XCHAR\* Path, /\* 驱动器的根目录 \*/

DWORD\* Clusters, /\* 存储空闲簇数目变量的指针 \*/

FATFS\*\* FileSystemObject /\* 文件系统对象指针的指针 \*/

);

参数

Path'\0'结尾的字符串指针，该字符串指定了逻辑驱动器的目录。

Clusters存储空闲簇数目的DWORD变量的指针。

FileSystemObject相应文件系统对象指针的指针。

返回值

FR\_OK (0)函数成功。\*Clusters表示空闲簇的数目，并且\*FileSystemObject指向文件系统对象。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_getfree函数当\_FS\_READONLY == 0并且\_FS\_MINIMIZE == 0时有效。

f\_getfree函数获取驱动器上空闲簇的数目。文件系统对象中的成员csize是每簇中的扇区数，因此，以扇区为单位的空闲空间可以被计算出来。当FAT32卷上的FSInfo结构不同步时，该函数返回一个错误的空闲簇计数。

列子

FATFS \*fs;

DWORD fre\_clust, fre\_sect, tot\_sect;

/\* Get drive information and free clusters \*/

res = f\_getfree("/", &fre\_clust, &fs);

if (res) die(res);

/\* Get total sectors and free sectors \*/

tot\_sect = (fs->max\_clust - 2) \* fs->csize;

fre\_sect = fre\_clust \* fs->csize;

/\* Print free space in unit of KB (assuming 512B/sector) \*/

printf("%lu KB total drive space.\n"

"%lu KB available.\n",

fre\_sect / 2, tot\_sect / 2);

**f\_stat**

获取文件状态

FRESULT f\_stat (

const XCHAR\* FileName, /\* 文件名或目录名的指针 \*/

FILINFO\* FileInfo /\* FILINFO结构的指针 \*/

);

参数

FileName'\0'结尾的字符串指针，该字符串指定了待获取其信息的文件或目录。

FileInfo存储信息的空白FILINFO结构的指针。

返回值

FR\_OK (0)函数成功。

FR\_NO\_FILE找不到文件或目录。

FR\_NO\_PATH找不到路径。

FR\_INVALID\_NAME路径名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_stat函数当\_FS\_MINIMIZE == 0时可用。  
f\_stat函数获取一个文件或目录的信息。信息的详情，请参考FILINFO结构和f\_readdir函数。

**f\_mkdir**

创建一个目录

FRESULT f\_mkdir (

const XCHAR\* DirName /\* 目录名的指针 \*/

);

参数

DirName'\0'结尾的字符串指针，该字符串指定了待创建的目录名。

返回值

FR\_OK (0)函数成功。

FR\_NO\_PATH找不到路径。

FR\_INVALID\_NAME路径名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_DENIED由于目录表或磁盘满，而导致目录不能被创建。

FR\_EXIST已经存在同名的文件或目录。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_WRITE\_PROTECTED存储介质被写保护。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_mkdir函数当\_FS\_READONLY == 0并且\_FS\_MINIMIZE == 0时可用。  
f\_mkdir函数创建一个新目录。

例子

res = f\_mkdir("sub1");

if (res) die(res);

res = f\_mkdir("sub1/sub2");

if (res) die(res);

res = f\_mkdir("sub1/sub2/sub3");

if (res) die(res);

**f\_unlink**

移除一个对象

FRESULT f\_unlink (

const XCHAR\* FileName /\* 对象名的指针 \*/

);

参数

FileName'\0'结尾的字符串指针，该字符串指定了一个待移除的对象。

返回值

FR\_OK (0)函数成功。

FR\_NO\_FILE找不到文件或目录。

FR\_NO\_PATH找不到路径。

FR\_INVALID\_NAME路径名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_DENIED由于下列原因之一，而导致该函数被拒绝：

* 对象具有只读属性
* 目录不是空的

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_WRITE\_PROTECTED存储介质被写保护。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_unlink函数当\_FS\_READONLY == 0并且\_FS\_MINIMIZE == 0时可用。  
f\_unlink函数移除一个对象。不要移除打开的对象或当前目录。

**f\_chmod**

修改一个文件或目录的属性。

FRESULT f\_chmod (

const XCHAR\* FileName, /\* 文件或目录的指针 \*/

BYTE Attribute, /\* 属性标志 \*/

BYTE AttributeMask /\* 属性掩码 \*/

);

参数

FileName'\0'结尾的字符串指针，该字符串指定了一个待被修改属性的文件或目录。

Attribute待被设置的属性标志，可以是下列标志的一个或任意组合。指定的标志被设置，其他的被清除。

|  |  |
| --- | --- |
| **属性** | **描述** |
| **AM\_RDO** | 只读 |
| **AM\_ARC** | 存档 |
| **AM\_SYS** | 系统 |
| **AM\_HID** | 隐藏 |

AttributeMask属性掩码，指定修改哪个属性。指定的属性被设置或清除。

返回值

FR\_OK (0)函数成功。

FR\_NO\_FILE找不到文件或目录。

FR\_NO\_PATH找不到路径。

FR\_INVALID\_NAME路径名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_WRITE\_PROTECTED存储介质被写保护。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_chmod函数当\_FS\_READONLY == 0并且\_FS\_MINIMIZE == 0时可用。  
f\_chmod函数修改一个文件或目录的属性。

例子

// 设置只读标志，清除存档标志，其他不变

f\_chmod("file.txt", AR\_RDO, AR\_RDO | AR\_ARC);

**f\_utime**

f\_utime函数修改一个文件或目录的时间戳。

FRESULT f\_utime (

const XCHAR\* FileName, /\* 文件或目录路径的指针 \*/

const FILINFO\* TimeDate /\* 待设置的时间和日期 \*/

);

参数

FileName'\0'结尾的字符串的指针，该字符串指定了一个待修改时间戳的文件或目录。

TimeDate文件信息结构指针，其中成员ftime和fdata存储了一个待被设置的的时间戳。不关心任何其他成员。

返回值

FR\_OK (0)函数成功。

FR\_NO\_FILE找不到文件或目录。

FR\_NO\_PATH找不到路径。

FR\_INVALID\_NAME路径名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_WRITE\_PROTECTED存储介质被写保护。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_utime函数当\_FS\_READONLY == 0并且\_FS\_MINIMIZE == 0时可用。  
f\_utime函数修改一个文件或目录的时间戳。

**f\_rename**

重命名一个对象。

FRESULT f\_rename (

const XCHAR\* OldName, /\* 原对象名的指针 \*/

const XCHAR\* NewName /\* 新对象名的指针 \*/

);

参数

OldName'\0'结尾的字符串的指针，该字符串指定了待被重命名的原对象名。

NewName'\0'结尾的字符串的指针，该字符串指定了重命名后的新对象名，不能包含驱动器号。

返回值

FR\_OK (0)函数成功。

FR\_NO\_FILE找不到原名。

FR\_NO\_PATH找不到路径。

FR\_INVALID\_NAME文件名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_EXIST新名和一个已存在的对象名冲突。

FR\_DENIED由于任何原因，而导致新名不能被创建。

FR\_WRITE\_PROTECTED存储介质被写保护。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_rename函数当\_FS\_READONLY == 0并且\_FS\_MINIMIZE == 0时可用。  
f\_rename函数重命名一个对象，并且也可以将对象移动到其他目录。逻辑驱动器号由原名决定，新名不能包含一个逻辑驱动器号。不要重命名打开的对象。

例子

/\* 重命名一个对象 \*/

f\_rename("oldname.txt", "newname.txt");

/\* 重命名并且移动一个对象到另一个目录 \*/

f\_rename("oldname.txt", "dir1/newname.txt");

**f\_mkfs**

在驱动器上创建一个文件系统

FRESULT f\_mkfs (

BYTE Drive, /\* 逻辑驱动器号 \*/

BYTE PartitioningRule, /\* 分区规则 \*/

WORD AllocSize /\* 分配单元大小 \*/

);

参数

Drive待格式化的逻辑驱动器号(0-9)。

PartitioningRule当给定0时，首先在驱动器上的第一个扇区创建一个分区表，然后文件系统被创建在分区上。这被称为FDISK格式化，用于硬盘和存储卡。当给定1时，文件系统从第一个扇区开始创建，而没有分区表。这被称为超级软盘(SFD)格式化，用于软盘和可移动磁盘。

AllocSize指定每簇中以字节为单位的分配单元大小。数值必须是0或从512到32K之间2的幂。当指定0时，簇大小取决于卷大小。

返回值

FR\_OK (0)函数成功。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_WRITE\_PROTECTED驱动器被写保护。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_MKFS\_ABORTED由于下列原因之一，而导致函数在开始格式化前终止：

* 磁盘容量太小
* 参数无效
* 该驱动器不允许的簇大小。

描述

f\_mkfs函数当\_FS\_READOLNY == 0并且\_USE\_MKFS == 1时可用。  
f\_mkfs函数在驱动器中创建一个FAT文件系统。对于可移动媒介，有两种分区规则：FDISK和SFD，通过参数PartitioningRule选择。FDISK格式在大多数情况下被推荐使用。该函数当前不支持多分区，因此，物理驱动器上已存在的分区将被删除，并且重新创建一个占据全部磁盘空间的新分区。  
根据Microsoft发布的FAT规范，FAT分类：FAT12/FAT16/FAT32，由驱动器上的簇数决定。因此，选择哪种FAT分类，取决于卷大小和指定的簇大小。簇大小影响文件系统的性能，并且大簇会提高性能。

**f\_forward**

读取文件数据并将其转发到数据流设备。

FRESULT f\_forward (

FIL\* FileObject, /\* 文件对象 \*/

UINT (\*Func)(const BYTE\*,UINT), /\* 数据流函数 \*/

UINT ByteToFwd, /\* 要转发的字节数 \*/

UINT\* ByteFwd /\* 已转发的字节数 \*/

);

参数

FileObject打开的文件对象的指针。

Func用户定义的数据流函数的指针。详情参考示例代码。

ByteToFwd要转发的字节数，UINT范围内。

ByteFwd返回已转发的字节数的UINT变量的指针。

返回值

FR\_OK (0)函数成功。

FR\_DENIED由于文件已经以非读模式打开，而导致函数失败。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_INVALID\_OBJECT文件对象无效。

描述

f\_forward函数当\_USE\_FORWARD == 1并且\_FS\_TINY == 1时可用。  
f\_forward函数从文件中读取数据并将数据转发到输出流，而不使用数据缓冲区。这适用于小存储系统，因为它在应用模块中不需要任何数据缓冲区。文件对象的文件指针以转发的字节数增加。如果\*ByteFwd < ByteToFwd并且没有错误，则意味着由于文件结束或在数据传输过程中流忙，请求的字节不能被传输。

例子(音频播放)

/\*-----------------------------------------------------------------------\*/

/\* 示例代码：数据传输函数，将被f\_forward函数调用 \*/

/\*-----------------------------------------------------------------------\*/

UINT out\_stream ( /\* 返回已发送字节数或流状态 \*/

const BYTE \*p, /\* 将被发送的数据块的指针 \*/

UINT btf /\* >0: 传输调用(将被发送的字节数)。0: 检测调用 \*/

)

{

UINT cnt = 0;

if (btf == 0) { /\* 检测调用 \*/

/\* 返回流状态(0: 忙，1: 就绪) \*/

/\* 当检测调用时，一旦它返回就绪，那么在后续的传输调用时，它必须接收至少一个字节，或者f\_forward将以FR\_INT\_ERROR而失败。 \*/

if (FIFO\_READY) cnt = 1;

}

else { /\* 传输调用 \*/

do { /\* 当有数据要发送并且流就绪时重复 \*/

FIFO\_PORT = \*p++;

cnt++;

} while (cnt < btf && FIFO\_READY);

}

return cnt;

}

/\*-----------------------------------------------------------------------\*/

/\* 示例代码：使用f\_forward函数 \*/

/\*-----------------------------------------------------------------------\*/

FRESULT play\_file (

char \*fn /\* 待播放的音频文件名的指针 \*/

)

{

FRESULT rc;

FIL fil;

UINT dmy;

/\* 以只读模式打开音频文件 \*/

rc = f\_open(&fil, fn, FA\_READ);

/\* 重复，直到文件指针到达文件结束位置 \*/

while (rc == FR\_OK && fil.fptr < fil.fsize) {

/\* 任何其他处理... \*/

/\* 定期或请求式填充输出流 \*/

rc = f\_forward(&fil, out\_stream, 1000, &dmy);

}

/\* 该只读的音频文件对象不需要关闭就可以被丢弃 \*/

return rc;

}

**f\_chdir**

f\_chdir函数改变一个驱动器的当前目录。

FRESULT f\_chdir (

const XCHAR\* Path /\* 路径名的指针 \*/

);

参数

Path'\0'结尾的字符串的指针，该字符串指定了将要进去的目录。

返回值

FR\_OK (0)函数成功。

FR\_NO\_PATH找不到路径。

FR\_INVALID\_NAME路径名无效。

FR\_INVALID\_DRIVE驱动器号无效。

FR\_NOT\_READY由于驱动器中没有存储介质或任何其他原因，而导致磁盘驱动器无法工作。

FR\_DISK\_ERR由于底层磁盘I/O函数中的错误，而导致该函数失败。

FR\_INT\_ERR由于一个错误的FAT结构或一个内部错误，而导致该函数失败。

FR\_NOT\_ENABLED逻辑驱动器没有工作区。

FR\_NO\_FILESYSTEM磁盘上没有有效的FAT卷。

描述

f\_chdir函数当\_FS\_RPATH == 1时可用。  
f\_chdir函数改变一个逻辑驱动器的当前目录。当一个逻辑驱动器被自动挂载时，它的当前目录被初始化为根目录。注意：当前目录被保存在每个文件系统对象中，因此它也影响使用同一逻辑驱动器的其它任务。

例子

// 改变当前驱动器的当前目录(根目录下的dir1)

f\_chdir("/dir1");

// 改变驱动器2的当前目录(父目录)

f\_chdir("2:..");

**f\_chdrive**

f\_chdrive函数改变当前驱动器。

FRESULT f\_chdrive (

BYTE Drive /\* 逻辑驱动器号 \*/

);

Drive指定将被设置为当前驱动器的逻辑驱动器号。

返回值

FR\_OK (0)函数成功。

FR\_INVALID\_DRIVE驱动器号无效。

描述

f\_chdrive函数当\_FS\_RPATH == 1时可用。  
f\_chdrive函数改变当前驱动器。当前驱动器号初始值为0，注意：当前驱动器被保存为一个静态变量，因此它也影响使用文件函数的其它任务。

**f\_gets**

f\_gets从文件中读取一个字符串。

char\* f\_gets (

char\* Str, /\* 读缓冲区 \*/

int Size, /\* 读缓冲区大小 \*/

FIL\* FileObject /\* 文件对象 \*/

);

参数

Str存储读取字符串的读缓冲区指针。

Size读缓冲区大小。

FileObject打开的文件对象结构指针。

返回值当函数成功后，Str将被返回。

描述

f\_gets函数当\_USE\_STRFUNC == 1或者\_USE\_STRFUNC == 2时可用。如果\_USE\_STRFUNC == 2，文件中包含的'\r'则被去除。  
f\_gets函数是f\_read的一个封装函数。当读取到'\n'、文件结束或缓冲区被填冲了Size - 1个字符时，读操作结束。读取的字符串以'\0'结束。当文件结束或读操作中发生了任何错误，f\_gets()返回一个空字符串。可以使用宏f\_eof()和f\_error()检查EOF和错误状态。

**f\_putc**

f\_putc函数向文件中写入一个字符。

int f\_putc (

int Chr, /\* 字符 \*/

FIL\* FileObject /\* 文件对象 \*/

);

参数

Chr待写入的字符。

FileObject打开的文件对象结构的指针。

返回值

当字符被成功地写入后，函数返回该字符。由于磁盘满或任何错误而导致函数失败，将返回EOF。

描述

f\_putc函数当(\_FS\_READONLY == 0)&&(\_USE\_STRFUNC == 1 || \_USE\_STRFUNC == 2)时可用。当\_USE\_STRFUNC == 2时，字符'\n'被转换为"\r\n"写入文件中。  
f\_putc函数是f\_write的一个封装函数。

**f\_puts**

f\_puts函数向文件中写入一个字符串。

int f\_puts (

const char\* Str, /\* 字符串指针 \*/

FIL\* FileObject /\* 文件对象指针 \*/

);

参数

Str待写入的'\0'结尾的字符串的指针。'\0'字符不会被写入。

FileObject打开的文件对象结构的指针。

返回值

函数成功后，将返回写入的字符数。由于磁盘满或任何错误而导致函数失败，将返回EOF。

描述

f\_puts()当(\_FS\_READONLY == 0)&&(\_USE\_STRFUNC == 1 || \_USE\_STRFUNC == 2)时可用。当\_USE\_STRFUNC == 2时，字符串中的'\n'被转换为"\r\n"写入文件中。  
f\_puts()是f\_putc()的一个封装函数。

**f\_printf**

f\_printf函数向文件中写入一个格式化字符串。

int f\_printf (

FIL\* FileObject, /\* 文件对象指针 \*/

const char\* Foramt, /\* 格式化字符串指针 \*/

...

);

参数

FileObject已打开的文件对象结构的指针。

Format'\0'结尾的格式化字符串指针。

...

可选参数

返回值

函数成功后，将返回写入的字符数。由于磁盘满或任何错误而导致函数失败，将返回EOF。

描述

f\_printf函数当(\_FS\_READONLY == 0)&&(\_USE\_STRFUNC == 1 || \_USE\_STRFUNC == 2)时可用。当\_USE\_STRFUNC == 2时，包含在格式化字符串中的'\n'将被转换成"\r\n"写入文件中。

f\_printf函数是f\_putc和f\_puts的一个封装函数。如下所示，格式控制符是标准库的一个子集：  
类型：c s d u X  
大小：l  
标志：0

例子

f\_printf(&fil, "%6d", -200); /\* " -200" \*/

f\_printf(&fil, "%02u", 5); /\* "05" \*/

f\_printf(&fil, "%ld", 12345678L); /\* "12345678" \*/

f\_printf(&fil, "%08lX", 1194684UL); /\* "00123ABC" \*/

f\_printf(&fil, "%s", "String"); /\* "String" \*/

f\_printf(&fil, "%c", 'a'); /\* "a" \*/

**磁盘I/O接口**

由于FatFs模块完全与磁盘I/O层分开，因此底层磁盘I/O需要下列函数去读/写物理磁盘以及获取当前时间。由于底层磁盘I/O模块并不是FatFs的一部分，因此它必须由用户提供。

**[[编辑](http://wiki.emsym.com/index.php?title=TB_FatFs%E7%A7%BB%E6%A4%8D%E5%AE%9E%E9%AA%8C&action=edit&section=32" \o "编辑段落：disk initialize)] disk\_initialize**

初始化磁盘驱动器

DSTATUS disk\_initialize (

BYTE Drive /\* 物理驱动器号 \*/

);

参数

Drive指定待初始化的物理驱动器号。

返回值

disk\_initialize函数返回一个磁盘状态作为结果。磁盘状态的详情，参考disk\_status函数。

描述

disk\_initialize函数初始化一个物理驱动器。函数成功后，返回值中的STA\_NOINIT标志被清除。

disk\_initialize函数被FatFs模块在卷挂载过程中调用，去管理存储介质的改变。**当FatFs模块起作用时，或卷上的FAT结构可以被瓦解时，应用程序不能调用该函数。可以使用f\_mount函数去重新初始化文件系统。**

**disk\_status**

获取当前磁盘的状态

DSTATUS disk\_status (

BYTE Drive /\* 物理驱动器号\*/

);

参数

Drive指定待确认的物理驱动器号。

返回值

磁盘状态，是下列标志的组合：STA\_NOINIT

指示磁盘驱动器还没有被初始化。当系统复位、磁盘移除和disk\_initialize函数失败时，该标志被设置；当disk\_initialize函数成功时，该标志被清除。STA\_NODISK

指示驱动器中没有存储介质。当安装了磁盘驱动器后，该标志始终被清除。

STA\_PROTECTED

指示存储介质被写保护。在不支持写保护缺口的驱动器上，该标志始终被清除。当STA\_NODISK被设置时，该标志无效。

**disk\_read**

从磁盘驱动器中读取扇区

DRESULT disk\_read (

BYTE Drive, /\* 物理驱动器号 \*/

BYTE\* Buffer, /\* 读取数据缓冲区的指针 \*/

DWORD SectorNumber, /\* 起始扇区号 \*/

BYTE SectorCount /\* 要读取的扇区数 \*/

);

参数

Drive指定物理驱动器号。

Buffer存储读取数据的缓冲区的指针。该缓冲区大小需要满足要读取的字节数(扇区大小 \* 扇区总数。由上层指定的存储器地址可能会也可能不会以字边界对齐。SectorNumber

指定在逻辑块地址(LBA)中的起始扇区号。

SectorCount指定要读取的扇区数(1-255)。

返回值

RES\_OK (0)函数成功

RES\_ERROR在读操作过程中发生了不能恢复的硬错误。

RES\_PARERR无效的参数。

RES\_NOTRDY磁盘驱动器还没被初始化。

**disk\_write**

向磁盘驱动器中写入扇区

DRESULT disk\_write (

BYTE Drive, /\* 物理驱动器号 \*/

const BYTE\* Buffer, /\* 写入数据缓冲区的指针(可能未对齐) \*/

DWORD SectorNumber, /\* 起始扇区号 \*/

BYTE SectorCount /\* 要写入的扇区数 \*/

);

参数

Drive指定物理驱动器号。

Buffer存储写入数据的缓冲区的指针。由上层指定的存储器地址可能会也可能不会以字边界对齐。

SectorNumber指定在逻辑块地址(LBA)中的起始扇区号。

SectorCount指定要写入的扇区数(1-255)。

返回值

RES\_OK (0)函数成功

RES\_ERROR在读操作过程中发生了不能恢复的硬错误。

RES\_WRPRT存储介质被写保护。

RES\_PARERR无效的参数。

RES\_NOTRDY磁盘驱动器还没被初始化。

描述

在只读配置中，不需要此函数。

**disk\_ioctl**

控制设备特定的功能以及磁盘读写以外的其它功能。

DRESULT disk\_ioctl (

BYTE Drive, /\* 驱动器号 \*/

BYTE Command, /\* 控制命令代码 \*/

void\* Buffer /\* 数据传输缓冲区 \*/

);

参数

Drive指定驱动器号(1-9)。

Command指定命令代码。

Buffer取决于命令代码的参数缓冲区的指针。当不使用时，指定一个NULL指针。

返回值

RES\_OK (0)函数成功。

RES\_ERROR发生错误。

RES\_PARERR无效的命令代码。

RES\_NOTRDY磁盘驱动器还没被初始化。

描述

FatFs模块只使用下述与设备无关的命令，没有使用任何设备相关功能。

|  |  |
| --- | --- |
| **命令** | **描述** |
| **CTRL\_SYNC** | 确保磁盘驱动器已经完成等待写过程。当磁盘I/O模块有一个写回高速缓存时，立即冲洗脏扇区。在只读配置中，不需要该命令。 |
| **GET\_SECTOR\_SIZE** | 返回驱动器的扇区大小赋给Buffer指向的WORD变量。在单个扇区大小配置中(\_MAX\_SS 为 512)，不需要该命令。 |
| **GET\_SECTOR\_COUNT** | 返回总扇区数赋给Buffer指向的DWORD变量。只在f\_mkfs函数中，使用了该命令。 |
| **GET\_BLOCK\_SIZE** | 返回以扇区为单位的存储阵列的擦除块大小赋给Buffer指向的DWORD变量。当擦除块大小未知或是磁盘设备时，返回1。只在f\_mkfs函数中，使用了该命令。 |

**get\_fattime**

获取当前时间

DWORD get\_fattime (void);

返回值

返回的当前时间被打包进一个DWORD数值。各位域定义如下：

bit31:25年，从1980年开始算起(0..127)

bit24:21月(1..12)

bit20:16日(1..31)

bit15:11时(0..23)

bit10:5分(0..59)

bit4:0秒/2(0..29)，由此可见FatFs的时间分辨率为2秒

描述

get\_fattime函数必须返回任何有效的时间，即使系统不支持实时时钟。如果返回一个0，则文件将没有一个有效的时间。在只读配置中，不需要此函数。