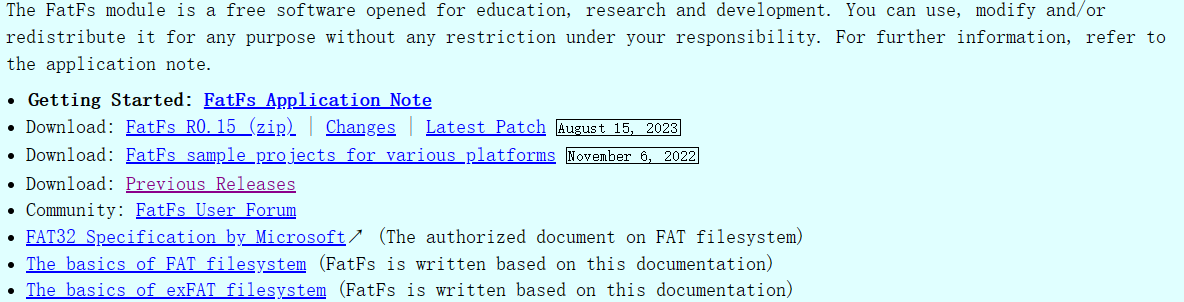
1. 下载源码

## 下载FatFs源码

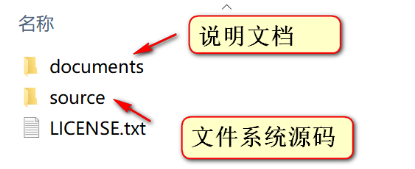
历史版本

最新版本

[FatFs - Generic FAT Filesystem Module](http://elm-chan.org/fsw/ff/00index_e.html" \o "FatFs - Generic FAT Filesystem Module)

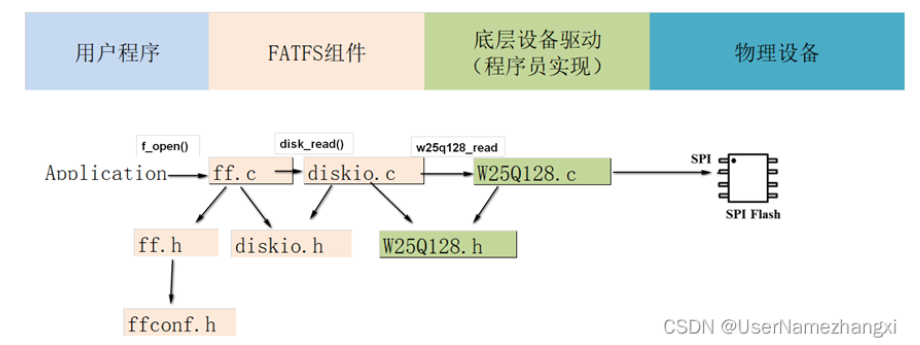


1. 认识FATFS源码结构



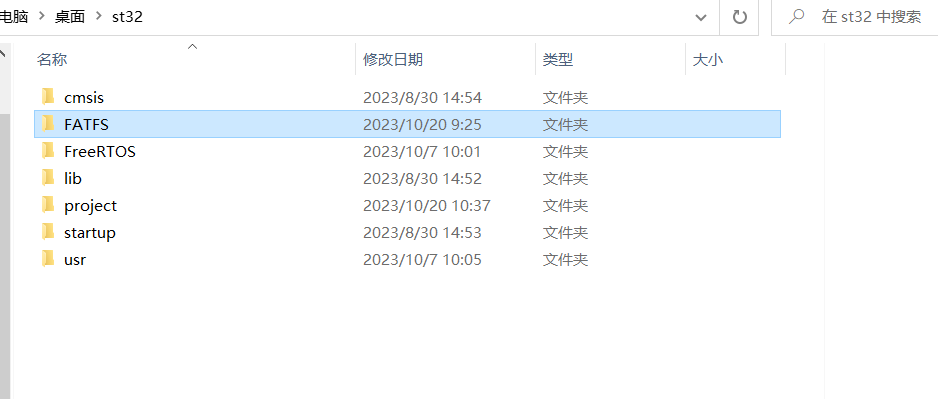




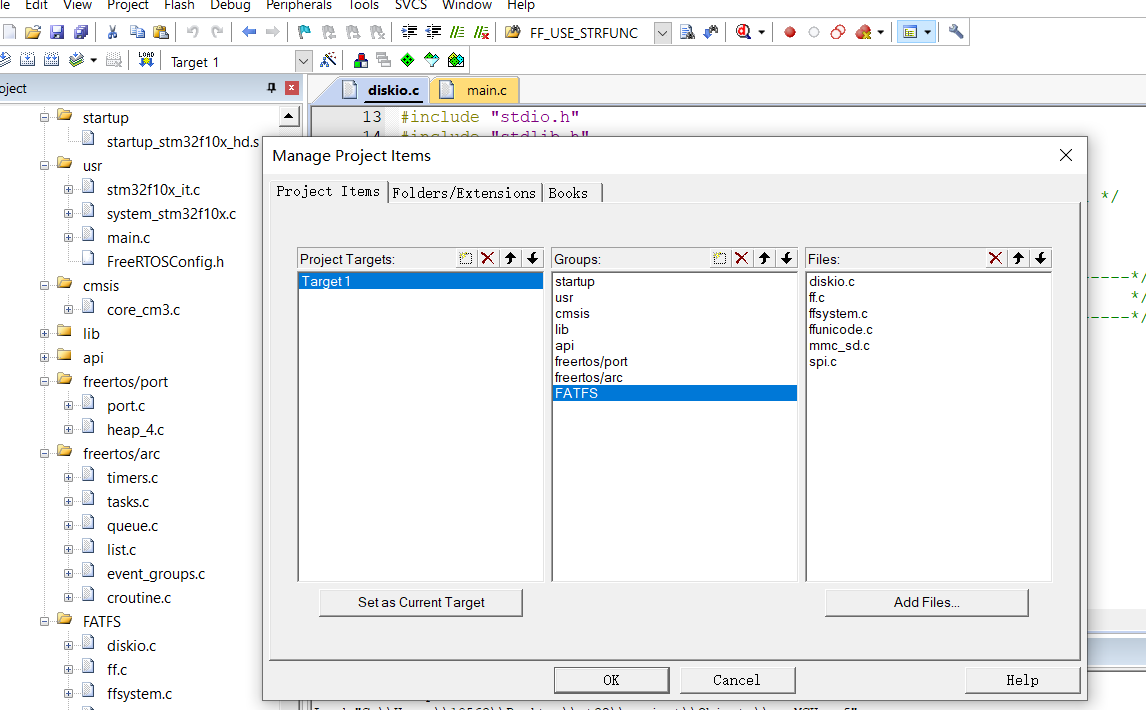


流程

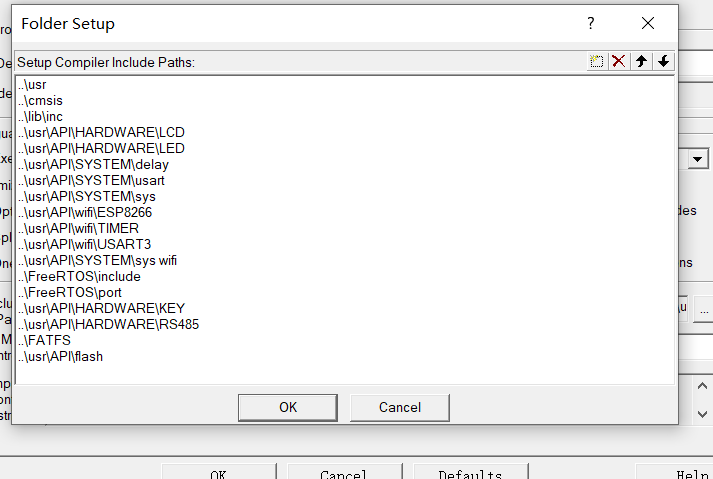
1. 移植添加文件





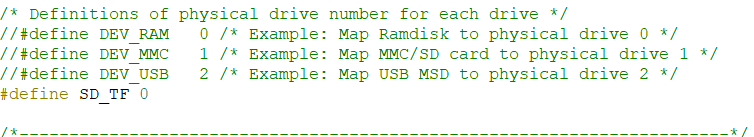


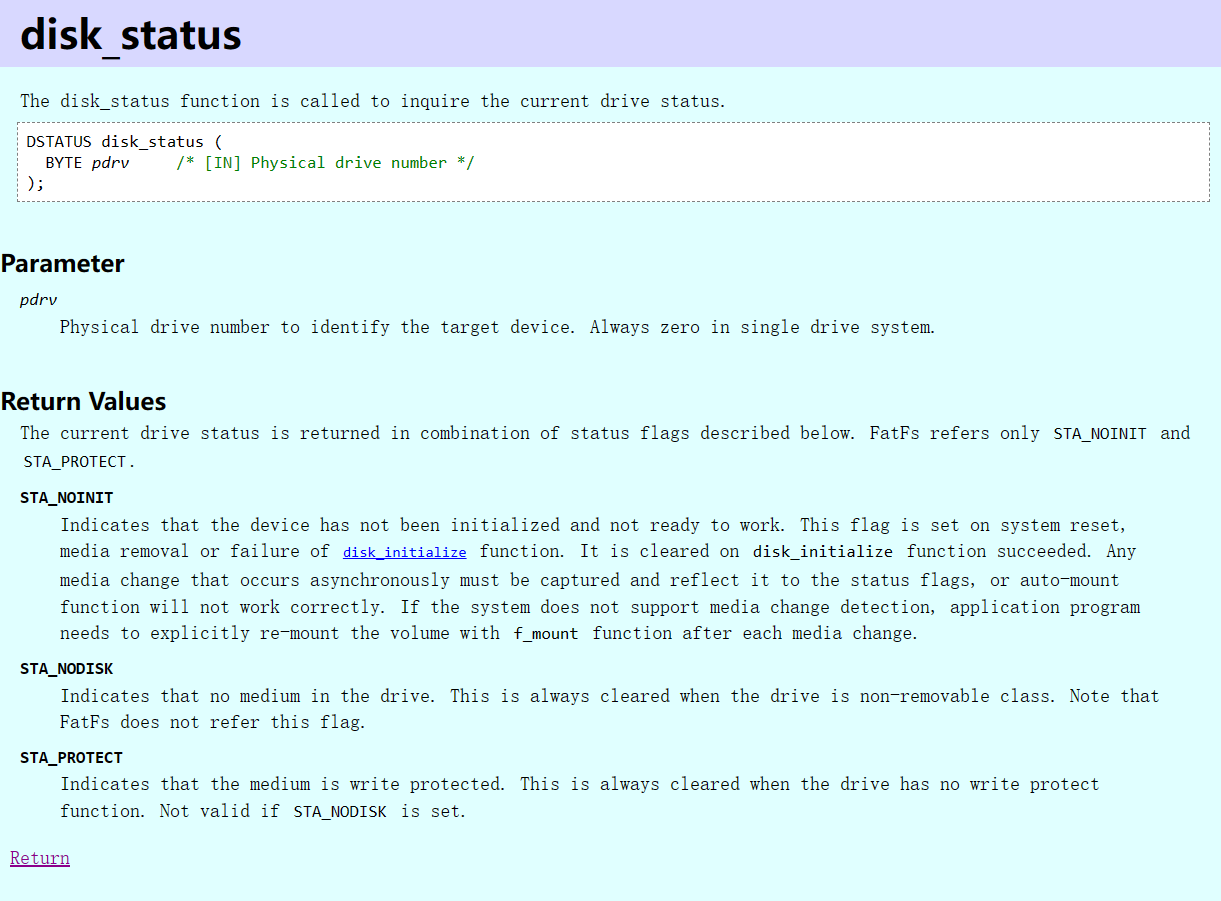
添加路径



1. 修改diskio.c文件

声明设备





官方对disk\_status()函数的解释。该函数只有一个形参，就是设备号用于标识不同的设备，在只有一个设备的情况下，设备号默认为0。

该函数的返回值有：

STA\_NOINIT：标识该设备未初始化成功，未进入就绪状态。

STA\_NODISK：FatFs不引用此标志。

STA\_PROTECT：表示该设备已进行了写保护。如果设置了STA\_NODISK，则无效。

0：表示该设备初始化成功。

只用一个设备sd卡直接return RES\_OK;就行，多个设备可以使用

switch (pdrv) {

case 设备1 :

;

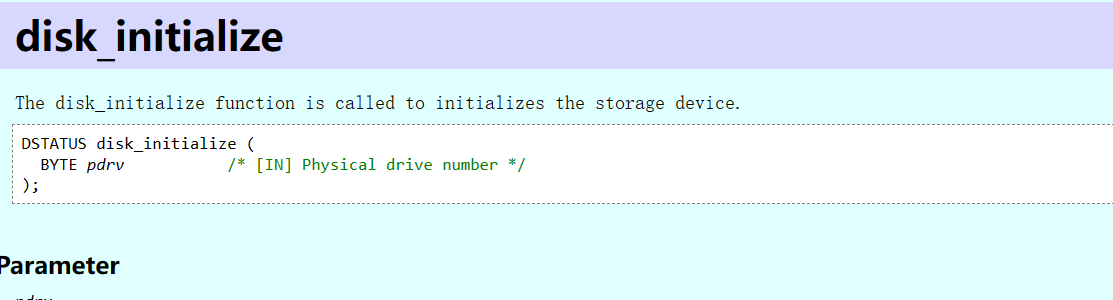
case 设备2 :

;

....

....

}



DSTATUS disk\_initialize (

BYTE pdrv /\* Physical drive nmuber to identify the drive \*/

)

{

Sd卡初始化

switch (pdrv) {

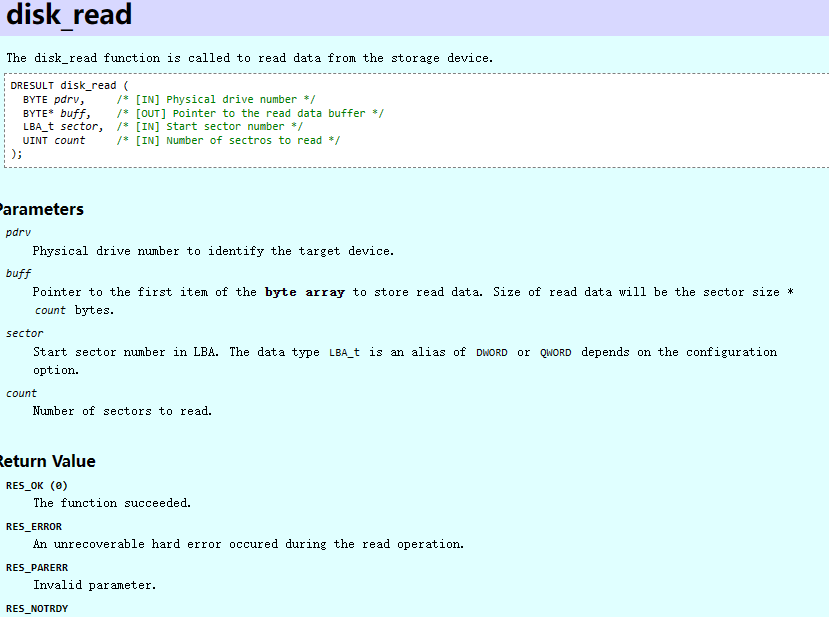
case SD\_TF :

return SD\_Config();

}

return STA\_NOINIT;

}



DRESULT disk\_read (

BYTE pdrv, /\* Physical drive nmuber to identify the drive \*/

BYTE \*buff, /\* Data buffer to store read data \*/

LBA\_t sector, /\* Start sector in LBA \*/

UINT count /\* Number of sectors to read \*/

)

{

switch (pdrv) {

case SD\_TF :

return (DRESULT)SD\_ReadDisk(buff, sector, count);

}

return RES\_PARERR;

}



DRESULT disk\_write (

BYTE pdrv, /\* Physical drive nmuber to identify the drive \*/

const BYTE \*buff, /\* Data to be written \*/

LBA\_t sector, /\* Start sector in LBA \*/

UINT count /\* Number of sectors to write \*/

)

{

switch (pdrv) {

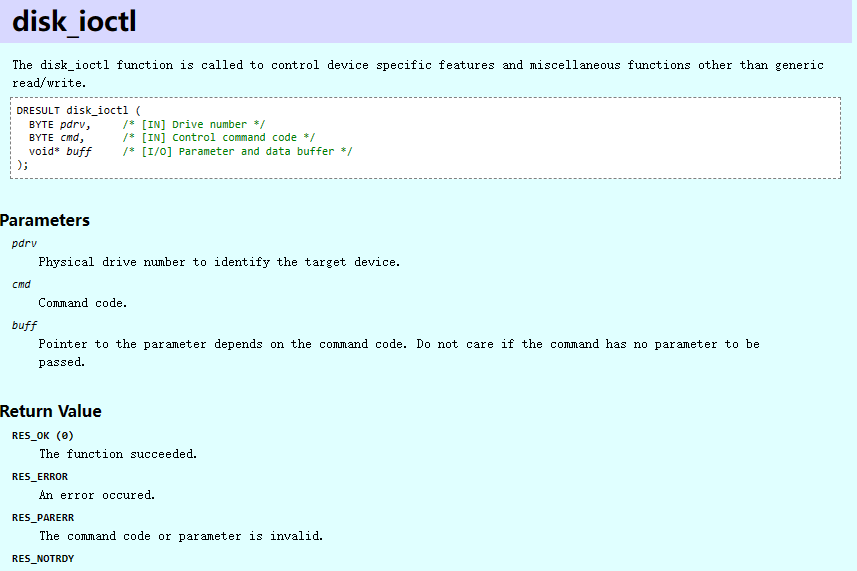
case SD\_TF :

return (DRESULT)SD\_WriteDisk((uint8\_t \*)buff, sector, count);

}

return RES\_PARERR;

}



DRESULT disk\_ioctl (

BYTE pdrv, /\* Physical drive nmuber (0..) \*/

BYTE cmd, /\* Control code \*/

void \*buff /\* Buffer to send/receive control data \*/

)

{

u8 result = RES\_OK;

switch (pdrv)

{

case SD\_TF :

switch(cmd)

{

/\* 完成挂起的写进程 \*/

case CTRL\_SYNC:

SD\_CS(0);

if(SD\_WaitReady())result = RES\_ERROR;

SD\_CS(1);

break;

/\* 扇区大小 \*/

case GET\_SECTOR\_SIZE:

\*(WORD \*)buff = 512;

break;

/\* 同时擦除扇区个数 \*/

case GET\_BLOCK\_SIZE:

\*(WORD \*)buff = 8;

break;

/\* 扇区数量\*/

case GET\_SECTOR\_COUNT:

\*(DWORD \*)buff = SD\_GetSectorCount();

break;

default:

result = RES\_PARERR;

break;

}

break;

}

if(!result)return RES\_OK;

return RES\_PARERR;

}



DWORD get\_fattime (void)

{

u32 date;

date =

(

((2015 - 1980) << 25) |

(7 << 21 ) |

(9 << 16 ) |

(12 << 11 ) |

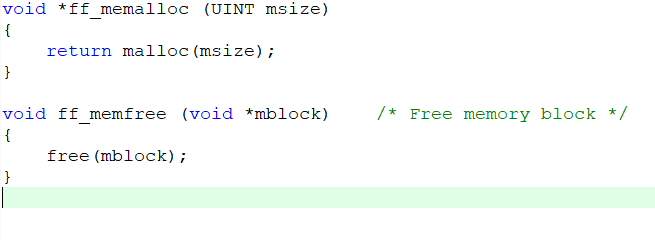
( 4 << 5 ) |

( 0 )

);

return date;

}



动态内存申请

  该文件中的第一部分函数就是动态内存申请接口。当然，其受制于在ffconf.h中的配置项FF\_USE\_LFN的限制。也就说，如果不使用长文件名，则 FatFs 不需要使用动态内存申请。 标准接口和标准C语言类似，就是名字稍有变化，有以下两个：

void\* ff\_memalloc (UINT msize); 申请动态内存

msize： 要申请的内存的大小

返回值：成功，返回指向申请的内存的指针；失败返回空指针

void ff\_memfree (void\* mblock); 释放之前申请的动态内存

mblock： 之前申请的动态内存的指针

返回值： 无

  为什么要自定义这两个函数？在使用了嵌入式操作系统之后，嵌入式操作系统一般都会提供对内存的管理功能。其一般也会重新定义动态内存的相关函数，而不是直接使用标准 C 语言定义的 malloc 和 free 函数。

void \*ff\_memalloc (UINT msize)

{

return malloc(msize);

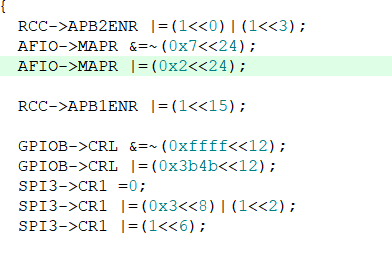
}

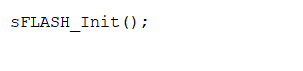
void ff\_memfree (void \*mblock) /\* Free memory block \*/

{

free(mblock);

}





管脚初始化

1. 测试

在main.c中加入

printf("SD\r\n");

if(SD\_Config() == 0)

{

printf("SD config OK\r\n");

if(f\_mount(&SD\_Fatfs,"0:",0) == FR\_OK)

{

if(f\_open(&fp,"0:5454564.txt",FA\_OPEN\_ALWAYS|FA\_READ|FA\_WRITE) == FR\_OK)

{

f\_write(&fp,"2023.07.17 为锁欲为组",sizeof("2023.07.17 为锁欲为组"),&bw);

printf("FATFS OK\r\n");

f\_close(&fp);

}

}

}

else

printf("SD config error\r\n");