```c
/*
*Description:文件操作等相关功能
*<br/>Copyright (C),2014-2015, minphone.linails
*<br/>This program is protected by copyright laws.
*<br/>Progarm Name:
*<br/>Date:2014/4/14
*<br/>Last modified Date:2014/4/30
*@author minphone.linails  linails@foxmail.com
*@version 1.0
*/
#include <stdio.h>
#include "MacroAndConst.h"
#include "fatapp.h"
#include "string.h"
#include "malloc.h"
#include "ff.h"

/**
  * @brief  文件类型列表
  */
const uint8 *type[8][9]=
{
    {"bin"},            /*BIN文件*/
    {"lrc"},            /*LRC文件*/
    {"nes"},            /*NES文件*/
    {"txt","c","h","doc","docx","pdf","exe","rar","chm"},       /*文本文件*/
    {"mp3","mp4","m4a","3gp","acc","wma","wav","mid","flac"},   /*音乐文件*/
    {"bmp","jpg","jpeg","gif"},                                 /*图片文件*/
    {"mmdb"},           /*机器学习数据库文件*/
    {"ntb"},            /*节点信息表文件    */
};

/**
  * @brief  定义了全局相关的文件系统变量
  */
 FATFS fs;          /*逻辑磁盘工作区*/
 FIL file;          /*文件          */
 FIL filetemp;      /*暂存文件      */
 UINT br,bw;        /*读写变量      */
 FILINFO fileinfo;  /*文件状态信息  */
 DIR dir;           /*目录          */

 uint8 fname_t[20][10]={0};     /*文件名暂存                */
 uint8 fatbuf[512];             /*SD卡数据缓存区            */
 uint8 mmdbDir[]="MacLMdb/";    /*机器学习数据库文件的路径名*/

/**
  * @fun    FileDirCreat
  * @brief  创建文件夹
  * @param  dirName         :指向打开文件名对象结构体的指针
  * @retval FRESULT         :返回值
  */
FRESULT FileDirCreat(const TCHAR *dirName)
{
    FRESULT res;
```

```
    f_mount(0,&fs);
    res=f_mkdir(dirName);
    f_mount(0,0);
    return res;
}

/**
  * @fun    FileRead_Top
  * @brief  读文件函数，文件存在的时候，直接打开进行写操作
  * @brief  文件不存在的时候，返回错误，可以读取大于512字节数据
  * @brief  从文件头开始读数据<使用缓冲区，执行速度快>
  * @param  fileName    :指向打开文件对象结构体的指针
  * @param  buffer      :指向要读出的数据缓冲区的指针
  * @param  length      :要读取的数据长度
  * @retval FRESULT     :返回值
  */
FRESULT FileRead_Top(const TCHAR *fileName, uint8 *buffer,uint32 length)
{
    FRESULT res;
    uint16  j=0;
    uint16  remainder=length%512;   /*求余*/
    f_mount(0,&fs);
    res=f_open(&file,fileName, FA_OPEN_EXISTING|FA_READ);
    if(res) return res;             /*错误返回*/
    else{
        if(length > f_size(&file)){ /*防止读取长度超过文件大小错误发生*/
            length=f_size(&file);
            remainder=length%512;   /*求余*/
        }
        /*读取大于等于512字节的倍数时的读取*/
        for(j=0;j<(length/512);j++){
            do{
                res=f_read(&file,fatbuf,512,&br);
                if(res) return res; /*错误返回*/
            }while(br!=512);
            strcpy((char *)&buffer[512*j],(const char *)fatbuf);
        }
        /*读取小于512字节时的读取*/
        if(remainder){
            do{
                res=f_read(&file,fatbuf,remainder,&br);
                if(res) return res; /*错误返回*/
            }while(br!=remainder);
            fatbuf[remainder+1]='\0';   /*在读取缓冲区末尾添加字符串结束标志*/
            strcpy((char *)&buffer[512*j],(const char *)fatbuf);
        }
        f_close(&file);             /*关闭文件*/
    }
    f_mount(0,0);
    return res;
}

/**
  * @fun    FileRead_pLocatLength
  * @brief  从指定位置读取指定长度的数据<使用缓冲区，执行速度快>
```

```
  * @param  RW_PointLength  :指向指定位置读写指定长度数据结构的指针
  * @retval FRESULT         :返回值
  */
FRESULT FileRead_pLocatLength(RW_PointLength_t *RW_PointLength)
{
    FRESULT res;
    uint16  j=0;
    uint16  remainder=(*RW_PointLength).length%512; /*求余*/
    f_mount(0,&fs);
    res=f_open(&file,RW_PointLength->fileName, FA_OPEN_EXISTING|FA_READ);
    if(res) return res;     /*错误返回*/
    else{
        /*防止读取长度超过文件大小发生错误*/
        if(((*RW_PointLength).length+(*RW_PointLength).offset) > f_size(&file)){
            (*RW_PointLength).length=f_size(&file)-(*RW_PointLength).offset;
            remainder=(*RW_PointLength).length%512; /*求余*/
        }
        res=f_lseek(&file,(*RW_PointLength).offset);
        if(res) return res; /*错误返回*/
        /*读取大于等于512字节的倍数时的读取*/
        for(j=0;j<((*RW_PointLength).length/512);j++){
            do{
                res=f_read(&file,fatbuf,512,&br);
                if(res) return res; /*错误返回*/
            }while(br!=512);
            strcpy((char *)&RW_PointLength->buffer[512*j],(const char *)fatbuf);
        }
        /*读取小于512字节时的读取*/
        if(remainder){
            do{
                res=f_read(&file,fatbuf,remainder,&br);
                if(res) return res;     /*错误返回*/
            }while(br!=remainder);
            fatbuf[remainder+1]='\0';   /*在读取缓冲区末尾添加字符串结束标志*/
            strcpy((char *)&RW_PointLength->buffer[512*j],(const char *)fatbuf);
        }
        f_close(&file);                 /*关闭文件*/
    }
    f_mount(0,0);
    return res;
}

/**
  * @fun    CreatFile
  * @brief  新建文件
  * @param  fileName    :指向打开文件对象结构体的指针
  * @retval FRESULT     :返回值
  */
FRESULT CreatFile(const TCHAR *fileName)
{
    FRESULT res;
    f_mount(0,&fs);
    res=f_open(&file,fileName, FA_CREATE_ALWAYS|FA_WRITE);
    if(res) return res;     /*错误返回*/
    else    f_close(&file); /*关闭文件*/
```

```
    f_mount(0,0);
    return res;
}

/**
  * @fun    DelFile
  * @brief  删除文件
  * @param  fileName    :指向打开文件对象结构体的指针
  * @retval FRESULT     :返回值
  */
FRESULT DelFile(const TCHAR *fileName)
{
    FRESULT res;
    f_mount(0,&fs);
    res=f_unlink((const TCHAR *)fileName);  /*删除文件*/
    f_mount(0,0);
    return res;
}

/**
  * @fun    FileWrite_Top
  * @brief  写文件函数，文件存在的时候，直接打开进行写操作
  * @brief  文件不存在的时候，先建立一个文件，然后进行写操作
  * @brief  从文件头开始写数据，即对文件进行刷新操作并覆盖
  * @param  fileName    :指向打开文件对象结构体的指针
  * @param  buffer      :指向要写入的数据缓冲区的指针
  * @retval FRESULT     :返回值
  */
FRESULT FileWrite_Top(const TCHAR *fileName,const uint8 *buffer)
{
    FRESULT res;
    uint16  size=0;
    f_mount(0,&fs);
    res=f_open(&file,fileName, FA_CREATE_ALWAYS|FA_WRITE);
    if(res) return res;         /*错误返回*/
    else{
        /*buffer 的数据长度*/
        size=strlen((const char *)buffer)+1;
        do{
            res=f_write(&file,buffer,size,&bw);
            if(res) return res; /*错误返回*/
        }while(bw!=size);
        f_close(&file);         /*关闭文件*/
    }
    f_mount(0,0);
    return res;
}

/**
  * @fun    FileWrite_replace
  * @brief  从指定位置替换指定长度的数据，即替换指定长度数据，即更新数据
  * @param  RW_PointLength  :指向指定位置读写指定长度数据结构的指针
  * @retval FRESULT         :返回值
  */
FRESULT FileWrite_replace(RW_PointLength_t *RW_PointLength)
```

```
{
    FRESULT res;
    uint16  size=0;
    f_mount(0,&fs);
    res=f_open(&file,RW_PointLength->fileName, FA_WRITE);
    if(res) return res;         /*错误返回*/
    else{
        /*要写入的数据长度*/
        size=(*RW_PointLength).length;
        res=f_lseek(&file,(*RW_PointLength).offset);
        if(res) return res;     /*错误返回*/
        do{
            res=f_write(&file,RW_PointLength->buffer,size,&bw);
            if(res) return res; /*错误返回*/
        }while(bw!=size);
        f_close(&file);         /*关闭文件*/
    }
    f_mount(0,0);
    return res;
}

/**
  * @fun    FileWrite_Add
  * @brief  从文件末尾添加数据
  * @param  RW_PointLength  :指向指定位置读写指定长度数据结构的指针
  * @retval FRESULT         :返回值
  */
FRESULT FileWrite_Add(RW_PointLength_t *RW_PointLength)
{
    FRESULT res;
    f_mount(0,&fs);
    res=f_open(&file,RW_PointLength->fileName, FA_WRITE);
    if(res) return res;         /*错误返回*/
    else{
        res=f_lseek(&file,file.fsize);
        if(res) return res;     /*错误返回*/
        do{
            res=f_write(&file,RW_PointLength->buffer,(*RW_PointLength).length,&bw);
            if(res) return res; /*错误返回*/
        }while(bw!=(*RW_PointLength).length);
        f_close(&file);         /*关闭文件*/
    }
    f_mount(0,0);
    return res;
}

/**
  * @fun    Clc_pLocatLength
  * @brief  擦除指定位置开始的指定长度的数据<使用缓冲区，执行速度快>
  * @param  RW_PointLength  :指向指定位置读写指定长度数据结构的指针
  * @retval FRESULT         :返回值
  */
FRESULT Clc_pLocatLength(RW_PointLength_t *RW_PointLength)
{
    FRESULT res;
```

```
    uint16  filesize;       /*文件大小      */
    uint16  fileremain;     /*文件剩余字节数*/
    uint8   clcflag=0xff;   /*删除标记      */
    uint16  i=0;
    f_mount(0,&fs);
    res=f_open(&file,RW_PointLength->fileName, FA_OPEN_EXISTING|FA_WRITE|FA_READ);
    if(res) return res;
    else{
        filesize=f_size(&file);/*获得文件大小*/
        do{
            /*获得文件剩余字节数*/
            fileremain=filesize-((*RW_PointLength).offset+(*RW_PointLength).length*(i+1));
            if(fileremain >= (*RW_PointLength).length){
                /*将读写指针指定到偏移量+数据长度处开始读取*/
                res=f_lseek(&file,(*RW_PointLength).offset+(*RW_PointLength).length*(i+1));
                if(res) return res;     /*错误返回*/
                do{
                    res=f_read(&file,fatbuf,(*RW_PointLength).length,&br);
                    if(res) return res; /*错误返回*/
                }while(br!=(*RW_PointLength).length);
                res=f_lseek(&file,(*RW_PointLength).offset+(*RW_PointLength).length*i);
                if(res) return res;     /*错误返回*/
                do{
                    res=f_write(&file,fatbuf,(*RW_PointLength).length,&bw);
                    if(res) return res; /*错误返回*/
                }while(bw!=(*RW_PointLength).length);
                if(f_tell(&file)==(filesize-(*RW_PointLength).length)){
                    res=f_truncate(&file);/*截断文件*/
                    if(res) return res;
                    else clcflag=0;
                }
                f_sync(&file);
            }else{
                /*将读写指针指定到偏移量+剩余数据长度处开始读取*/
                res=f_lseek(&file,(*RW_PointLength).offset+(*RW_PointLength).length*(i+1));
                if(res) return res;     /*错误返回*/
                do{
                    res=f_read(&file,fatbuf,fileremain,&br);
                    if(res) return res; /*错误返回*/
                }while(br!=fileremain);
                res=f_lseek(&file,(*RW_PointLength).offset+(*RW_PointLength).length*i);
                if(res) return res;     /*错误返回*/
                do{
                    res=f_write(&file,fatbuf,fileremain,&bw);
                    if(res) return res; /*错误返回*/
                }while(bw!=fileremain);
                if(f_tell(&file)==(filesize-(*RW_PointLength).length)){
                    res=f_truncate(&file);/*截断文件*/
                    if(res) return res;
                    else clcflag=0;
                }
            }
            i++;
            /*暂时不添加刷新缓冲数据*/
        }while(clcflag);
```

```
    }
    f_close(&file);
    f_mount(0,0);
    return res;
}

/**
  * @fun    FileInsert_pLocatLength
  * @brief  从指定位置开始插入指定长度的数据-路径名不能含驱动器号<使用缓冲区，执行速度快>
  * @param  RW_PointLength  :指向指定位置读写指定长度数据结构的指针
  * @retval FRESULT         :返回值
  */
FRESULT FileInsert_pLocatLength(RW_PointLength_t *RW_PointLength)
{
    FRESULT res;
    uint16  j;
    uint16  filesize;       /*文件大小      */
    uint16  fileremain;     /*文件剩余字节数*/
    uint16  remainder=(*RW_PointLength).offset%512;     /*求余      */
    uint8   *filename_tmp="FileInsert_pLocatLength.tmp";/*中间文件名*/

    f_mount(0,&fs);
    res=f_open(&filetemp,(const TCHAR*)filename_tmp, FA_CREATE_ALWAYS|FA_WRITE); if(res) ret
urn res;
    res=f_open(&file,(const TCHAR*)RW_PointLength->fileName, FA_OPEN_EXISTING|FA_READ); if(r
es) return res;
    if((*RW_PointLength).offset >= f_size(&file)){  /*防止插入位置超过文件大小错误发生*/
        (*RW_PointLength).offset=f_size(&file);
        remainder=(*RW_PointLength).offset%512;     /*求余*/
    }
    /*读取大于等于512字节的倍数时的读取*/
    for(j=0;j<((*RW_PointLength).offset/512);j++){
        do{
            res=f_read(&file,fatbuf,512,&br);
            if(res) return res; /*错误返回*/
        }while(br!=512);
        do{
            res=f_write(&filetemp,fatbuf,512,&bw);
            if(res) return res; /*错误返回*/
        }while(bw!=512);
    }
    /*读取小于512字节时的读取*/
    if(remainder){
        do{
            res=f_read(&file,fatbuf,remainder,&br);
            if(res) return res; /*错误返回*/
        }while(br!=remainder);
        do{
            res=f_write(&filetemp,fatbuf,remainder,&bw);
            if(res) return res; /*错误返回*/
        }while(bw!=remainder);
    }
    do{
        res=f_write(&filetemp,RW_PointLength->buffer,(*RW_PointLength).length,&bw);
        if(res) return res;     /*错误返回*/
```

```
    }while(bw!=(*RW_PointLength).length);
    filesize=f_size(&file);     /*获得文件大小*/
    fileremain=filesize-f_tell(&file);
    /*读取大于等于512字节的倍数时的读取*/
    for(j=0;j<(fileremain/512);j++){
        do{
            res=f_read(&file,fatbuf,512,&br);
            if(res) return res; /*错误返回*/
        }while(br!=512);
        do{
            res=f_write(&filetemp,fatbuf,512,&bw);
            if(res) return res; /*错误返回*/
        }while(bw!=512);
    }
    /*读取小于512字节时的读取*/
    if(fileremain%512){
        do{
            res=f_read(&file,fatbuf,fileremain%512,&br);
            if(res) return res; /*错误返回*/
        }while(br!=fileremain%512);
        do{
            res=f_write(&filetemp,fatbuf,fileremain%512,&bw);
            if(res) return res; /*错误返回*/
        }while(bw!=fileremain%512);
    }
    f_close(&file);
    f_close(&filetemp);
    f_unlink((const TCHAR *)RW_PointLength->fileName);  /*删除原文件*/
    /*路径名不能含驱动器号*/
    f_rename((const TCHAR *)filename_tmp,(const TCHAR *)RW_PointLength->fileName);/*重命名新
文件*/
    f_mount(0,0);
    return res;
}

/**
  * @fun    FileSwap_pLocatLength
  * @brief  将位置1开始的指定长度的数据和位置2开始的指定长度的数据进行位置调整
  * @param  FS_DataSwap     :指向指定位置读写指定长度数据结构的指针
  * @retval FRESULT         :返回值
  */
FRESULT FileSwap_pLocatLength(FS_DataSwap_t *FS_DataSwap)
{
    FRESULT res;
    uint16  i;
    uint8   cbuf_a;     /*一个字节缓冲  */
    uint8   cbuf_b;     /*一个字节缓冲  */
    UINT    br_a,bw_a;  /*读写变量      */
    UINT    br_b,bw_b;  /*读写变量      */

    f_mount(0,&fs);
    res=f_open(&file,FS_DataSwap->fileName, FA_WRITE|FA_READ); if(res) return res;
    else{
        for(i=0;i<(*FS_DataSwap).length;i++){   /*实现数据交换功能*/
            do{
```

```
                f_lseek(&file,(*FS_DataSwap).offset_loca+i);
                f_read(&file,&cbuf_a,1,&br_a);
                f_lseek(&file,(*FS_DataSwap).offset_locb+i);
                f_read(&file,&cbuf_b,1,&br_b);
            }while((br_a!=1)&&(br_b!=1));       /*要移动数据的读取*/
            do{
                f_lseek(&file,(*FS_DataSwap).offset_loca+i);
                f_write(&file,&cbuf_b,1,&bw_a);
                f_lseek(&file,(*FS_DataSwap).offset_locb+i);
                f_write(&file,&cbuf_a,1,&bw_b);
            }while((br_a!=1)&&(br_b!=1));       /*要移动数据的写入*/
        }
    }
    f_close(&file);
    f_mount(0,0);
    return res;
}

/**
  * @fun    SD_Getfree
  * @brief  获取sd卡使用信息
  * @param  drv     :指向空结尾字符串逻辑驱动器号的指针
  * @param  total   :总扇区数
  * @param  free    :总空闲扇区数
  * @retval FRESULT :返回值
  */
FRESULT SD_Getfree(uint8 *drv,uint32 *total,uint32 *free)
{
    FATFS *fs1; /*声明文件系统对象结构体的指针*/
    FRESULT res;
    DWORD fre_clust=0, fre_sect=0, tot_sect=0; /*定义空闲簇数、空闲扇区数、全部扇区数*/

    f_mount(0,&fs);
    res = f_getfree((const TCHAR*)drv, &fre_clust, &fs1);
    if(res) return res; /*错误返回*/
    else{
        tot_sect=(fs1->n_fatent-2)*fs1->csize;  /*得到总扇区数   */
        fre_sect=fre_clust*fs1->csize;          /*得到空闲扇区数*/
#if _MAX_SS!=512                                /*扇区大小不是512字节,则转换为512字节*/
        tot_sect*=fs1->ssize/512;
        fre_sect*=fs1->ssize/512;
#endif
        *total=tot_sect>>1; /*单位为KB*/
        *free=fre_sect>>1;  /*单位为KB*/
    }
    f_mount(0,0);
    return res;
}

/**
  * @fun    FileScan
  * @brief  path目录文件扫描，扫描出文件夹和文件
  * @param  path    :指向空结尾字符串目录名的指针
  * @retval FRESULT :返回值
  */
```

```
FRESULT FileScan(const uint8 *path)
{
    FRESULT res;
    uint8 lfn[30]="";   /*长文件名暂存*/
    uint8 *plfn=lfn;
#if _USE_LFN
    fileinfo.lfsize = sizeof(lfn);
    fileinfo.lfname = (TCHAR*)lfn;
#endif
    f_mount(0, &fs);
    res=f_opendir(&dir,(const TCHAR*)path);
    if(res) return res; /*错误返回*/
    else{
        while(1){
            res=f_readdir(&dir,&fileinfo);
            if(res) return res; /*错误返回*/
            else{
                if(!fileinfo.fname[0]) break;   /*空文件时跳出*/
                /*是文件夹(AM_DIR)或者是文件(AM_ARC)*/
                if((fileinfo.fattrib & AM_DIR)||(fileinfo.fattrib & AM_ARC)){
#if _USE_LFN
                    plfn = *fileinfo.lfname ? (uint8 *)fileinfo.lfname : (uint8 *)fileinfo.f
name;
#else
                    plfn = (uint8 *)fileinfo.fname;
#endif
                    printf("filename: %s\n",plfn);  /*测试时使用的打印语句*/
                }
            }
        }
    }
    f_mount(0,0);
    return res;
}

/**
  * @fun    SearchFileOrDir
  * @brief  对输入的文件或者文件夹进行查询存在与否，存在时num=1，不存在num=0
  * @param  name    :指向空结尾字符串目录名的指针
  * @param  num     :指向查找数量标记变量的指针
  * @retval FRESULT :返回值
  */
FRESULT SearchFileOrDir(const uint8 *name,uint8 *num)
{
    FRESULT res;
    f_mount(0, &fs);
    /*判断是文件还是文件夹*/
    if(strchr((const char*)name,'.')==NULL){    /*文件夹*/
        res=f_opendir(&dir,(const TCHAR*)name);
        if(res){
            *num=0;
            return res;     /*错误返回*/
        }else{
            *num=1;
        }
```

```
    }else{                  /*文件*/
        res=f_open(&file,(const TCHAR *)name, FA_OPEN_EXISTING|FA_READ);
        if(res){
            *num=0;
            return res;     /*错误返回*/
        }else{
            *num=1;
            f_close(&file); /*关闭文件*/
        }
    }
    f_mount(0,0);
    return res;
}

/**
  * @fun    GetFloder
  * @brief  路径获取
  * @param  dest    :指向空结尾目的字符串的指针
  * @param  name    :指向空结尾字符串文件名的指针
  * @retval None
  */
void GetFloder(uint8 *dest,const uint8 *name)
{
     uint8 i;
     uint8 *pAddr=(void*)0;
     pAddr=(uint8 *)strrchr((const char*)name,'/');
     i=(uint8)(pAddr-name);
     for(;i>0;i--){
        *dest++=*name++;
     }
}

/**
  * @fun    GenerateFilename
  * @brief  对路径和文件名进行组合生成含有路径的文件名
  * @param  dest    :指向目的字符串的指针
  * @param  folder  :指向空结尾字符串目录名的指针
  * @param  filename:指向空结尾字符串文件名的指针
  * @retval None
  */
void GenerateFilename(uint8 *dest,const uint8 *folder,const uint8 *filename)
{
    strcpy((char*)dest,(const char*)folder);
    dest[strlen((const char*)folder)]='/';
    strcpy((char*)&dest[strlen((const char*)folder)+1],(const char*)filename);
}
```