版本: **1.0.5 2014** 年 **3** 月

EXFAT 库接口说明

声　　明

联　系　方　式

安凯（广州）微电子技术有限公司

地址：广州科学城科学大道 182 号创新大厦 C1 区 3 楼
电话: (86)-20-3221 9000

传真: (86)-20-3221 9258

邮编: 510663

销售热线:

(86)-20-3221 9499

电子邮箱:

sales@anyka.com

主页:

http: //www. anyka.com

版 本 变 更 说 明

以下表格对于本文档的版本变更做一个简要的说明。版本变更仅限于技术内容的变更，不包括版式、格式、句法等的变更。

版本	说明	完成日期
V1.0.0	正式发布	2012 年 1 月
V1.0.1	修改增加打开文件时候的打开模式	2012 年 3 月
V1.0.2	1.增加二个接口，查找文件的接口，功能是查找到一个文件后就退出去， 2.给11BLUE增加回调，主要解决mount盘和播放提示音并行的问题。 3.增加二个通过打开文件夹的句柄进行查找文件的接口。 4.对 File_GetFileinfo 接口获取的 fdt 值进行附值，可通过此接口获取此文件的 fdt。	2013 年 3 月
V1.0.3	在 File_FindFirstEX 的接口上增加一个参数，以实现过滤不需要求找的文件夹	2013 年 4 月
V1.0.4	针对 10L/10C 平台增加一套查找功能的接口，以及增加一个针对文件夹下的文件的簇链修复的接口	2013 年 7 月
V1.0.5	针对 10L/10C 平台增加一套快速打开文件的接口，此接口只作特殊用途的。	2014 年 3 月

EXFAT 库接口说明与库对应的关系

Version	The Corresponding Lib
V0.0.1	Mono_Multi-task_V3.0.3
V0.0.2	Mono_Multi-task_V3.0.3
V0.0.3	Mono_Multi-task_V3.0.4
V0.0.4	EXFAT_V3.2.8
V0.0.5	EXFAT_V3.2.9
V0.0.6	EXFAT_V3.2.10
V0.0.7	EXFAT_V3.2.11
V1.0.0	EXFAT_V3.2.11
V1.0.1	EXFAT_V3.2.14
V1.0.2	EXFAT_V3.2.18_and_mount_v2.0.17
V1.0.3	EXFAT_V3.2.18_svn1594
V1.0.4	EXFAT_V3.2.24
V1.0.5	EXFAT_V3.2.30_and_mount_v2.0.21

目录

1 EXFAT库介绍

1.1 功能概述

EXFAT 项目是基于安凯多任务版本的文件系统进行二次开发，在原始版本的基础上增加了一些优化点，主要变更是对 FAT 管理、增加多任务的处理、引进 EXFAT 的支持。

1.2 使用范围

EXFAT 版本，很多模块都使用分步式加载，此版本某些功能只适应于多任务系统，而且对内存的使用以最合理的方式，对于内存较少的系统不一定能取得比较好的效果。

1.3 注意事项

暂无。

2 接口说明

2.1 文件列表搜索功能所需要的数据结构体

```
typedef struct tag_FileInfo
{
   T_U32 CreatTime;     //创建的时间
   T_U32 CreatDate;     //创建的日期
   T_U32 ModTime;       //修改的时间
   T_U32 ModDate;       //修改的日期
   T_U32 ParentId;      //父首簇号
   T_U32 FileId;        //文件首簇号
   T_U32 FileFdt;       //文件的 fdt
   T_U32 length_high;       //文件内容的大小
   T_U32 length_low;        //文件内容的大小
   T_U32 attr;          //文件的属性
   T_U32 NameLen;       // 长文件名的长度
   T_U16 LongName[300]; //文件的长文件名
   T_U8 ShortName[12];   //文件的短文件名，8.3 格式
}T_FILEINFO, *T_PFILEINFO;

    /* Following macros are used to filter files or folders when copying a folder or
      listing the files/folders in a folder. */
    #define FILTER_DEFAULT    0x0000

    // we will find it deeply, search all file include the sub folder
    #define FILTER_DEEP       0x0001

    // it will display all folder without matching, if it is set,FILTER_DEEP will be ignored
    #define FILTER_FOLDER     0x0002

    /* Control whether File_FindNext will iterate or not. Default is to iterate.
      When setting the macro, File_FindNext will not iterate. */
    #define FILTER_NOTITERATE   0x0004
```

```
typedef struct tag_FindBufCtrl
{
        T_U32 NodeCnt;   // 定义最大的缓存个数
        T_U8  DspCtrl;   // 是否显示 ".", ".." (DSP_PARENT | DSP_CURRENT)
        T_U8  patternLen; // 过滤条件的长度
        T_U16 type;      // 过滤的类型:例如 FILTER_DEEP.
        T_U16 *pattern;   //过滤的条件: 例如"*.mp3"; 查找所有的 mp3 文件
}T_FINDBUFCTRL, *T_PFINDBUFCTRL;


#define FILE_FIND_DEEP_CNT      7    //查找的最大层数不能超过 6 = 7 - 1
#define FIND_FILENAME_PATHLEN    262  //最大路径
struct tag_FileInfo_BeforePower
{
   T_EVERY_FILEINFO  EveryFileInfo[FILE_FIND_DEEP_CNT];   //记录每查找到一个文件的信息
   T_U32 deep_cnt;         //当前的层数
}T_FILEINFO_BEFOREPOWER, *T_PFILEINFO_BEFOREPOWER;

struct tag_FindBufInfo
{
   T_PFILEINFO_BEFOREPOWER find_fileinfo; //记录每一层文件的首簇号和 fdt
   T_U16 *pattern;      // need find
   T_U16 *fileter_pattern;  //no need find, only folder
   T_U16  find_type;   // SEARCH_TYPE_ITERATE or SEARCH_TYPE_NOTITERATE
   T_BOOL deep_flag;   //deep find or not

}T_FINDBUFINFO, *T_PFINDBUFINFO;

struct tag_FileInfo_Fastopen
{
   T_U32 ParentId;
   T_U32 FileId;
   T_U32 FileFdt;           // Record the corresponding the first  fdt of this file.
   T_U32 driver;            //driver
}T_FILEINFO_FASTOPEN, *T_PFILEINFO_FASTOPEN;
```

2.2 全局结构体信息

为 USB 部分 U 盘初始化提供一些标准结构体信息:

```
Typedef T_BOOL  (*F_DRIVER_Read) (T_U8 *buf,T_U32 BlkAddr, T_U32 BlkCnt,
T_U32 LunAddInfo);
Typedef T_BOOL (*F_DRIVER_Write) (T_U8 *buf,T_U32 BlkAddr, T_U32 BlkCnt,
T_U32 LunAddInfo);
Typedef struct _DRIVER_INFO_
{
        T_U16           nMainType;          // the value of E_MEDIUM
        T_U16           nSubType;           // the value of E_SUB_MEDIUM
        T_U32  nBlkSize;
        T_U32           nBlkCnt;
        F_DRIVER_Read       fRead;
        F_DRIVER_Write      fWrite;
} DRIVER _INFO, *PDRIVER _INFO;
```

在 U 盘方式,MOUNT 时,模块将输出这些结构体体信息,平台将根据这些信息再进行 U 盘 MOUNT 过程.

另外在 MOUNT 过程中,用户可能会选择 MOUNT 不同的分区,以下将让用户定义自己想 MOUNT 的分区类型.

```
typedef enum
{
  MEDIUM_RAM      = 0,
  MEDIUM_ROM      = 1,
  MEDIUM_NANDFLASH = 2,
  MEDIUM_SD       = 3,
  MEDIUM_NORFLASH  = 4,
  MEDIUM_DISKETTE  = 5,
  MEDIUM_FILE     = 6,
  MEDIUM_NANDRES   = 7,
  MEDIUM_USBHOST   = 8,
  MEDIUM_PARTITION = 9,
```

```
    MEDIUM_UNKNOWN   = 255
}E_MEDIUM;
Typedef enum {
        SYSTEM_PARTITION,        //系统分区,一般是保留分区,用户不可见
        USER_PARTITION,          //用户分区,此分区为 U 盘可见分区,
        ALL_PARTITION,           //所有分区.
}E_SUB_MEDIUM;
```

2.3 文件系统层的回调函数

```
typedef T_VOID (*F_OutStream)(T_U16 ch);
typedef T_U8 (*F_Instream)(T_VOID);
typedef T_U32 (*F_GetSecond)(T_VOID);
typedef T_VOID (*F_SetSecond)(T_U32 seconds);
/* Jan.10,07 - Modified to support Multi-Language */
typedef T_S32 (*F_UniToAsc)(const T_U16 *pUniStr, T_U32 UniStrLen,
                T_pSTR pAnsibuf, T_U32 AnsiBufLen, T_U32 code);
typedef T_S32 (*F_AscToUni)(const T_pSTR pAnsiStr, T_U32 AnsiStrLen,
                T_U16 *pUniBuf, T_U32 UniBufLen, T_U32 code);

typedef T_pVOID    (*F_RamAlloc)(T_U32 size, T_S8 *filename, T_U32 fileline);
typedef T_pVOID    (*F_RamRealloc)(T_pVOID var, T_U32 size, T_S8 *filename,
T_U32 fileline);
typedef T_pVOID (*F_RamFree)(T_pVOID var, T_S8 *filename, T_U32 fileline);

/* Sep.13,07 - Added to support Muti-Task. */
typedef T_S32 (*F_OsCrtSem)(T_U32 initial_count, T_U8 suspend_type, T_S8 *filename,
T_U32 fileline);
typedef T_S32 (*F_OsDelSem)(T_S32 semaphore, T_S8 *filename, T_U32 fileline);
typedef T_S32 (*F_OsObtSem)(T_S32 semaphore, T_U32 suspend, T_S8 *filename,
T_U32 fileline);
```

```
typedef T_S32 (*F_OsRelSem)(T_S32 semaphore, T_S8 *filename, T_U32 fileline);

typedef T_U32 (*F_GetChipID)(T_VOID);

typedef T_pVOID (*F_MemCpy)(T_pVOID dst, T_pCVOID src, T_U32 count);
typedef T_pVOID (*F_MemSet)(T_pVOID buf, T_S32 value, T_U32 count);
typedef T_pVOID (*F_MemMov)(T_pVOID dst, const T_pCVOID src, T_U32 count);
typedef int     (*F_MemCmp)(const T_pVOID buf1, const T_pCVOID buf2, T_U32 count);
typedef T_S32   (*F_Printf)(T_pCSTR s, ...);
typedef T_VOID (*F_MtdSysRst)(T_VOID);
typedef T_VOID (*F_MtdRandSeed)(T_VOID);
typedef T_U32 (*F_MtdGetRand)(T_U32);

typedef struct tag_FsInitInfo
{
   F_OutStream out;
   F_Instream  in;
   F_GetSecond fGetSecond;
   F_SetSecond fSetSecond;
   F_UniToAsc  fUniToAsc;
   F_AscToUni  fAscToUni;
   F_RamAlloc  fRamAlloc;
   F_RamRealloc fRamRealloc;
   F_RamFree  fRamFree;
   F_OsCrtSem fCrtSem;
   F_OsDelSem fDelSem;
   F_OsObtSem fObtSem;
   F_OsRelSem fRelSem;

   F_MemCpy  fMemCpy;
   F_MemSet  fMemSet;
   F_MemMov  fMemMov;
```

```
    F_MemCmp  fMemCmp;
    F_Printf  fPrintf;
    F_MtdSysRst    fMtdSysRst;
    F_MtdRandSeed  fMtdRandSeed;
    F_MtdGetRand  fMtdGetRand;

    F_GetChipID fGetChipId;
} T_FSCallback, * T_PFSCallback;
```

3 EXFAT接口说明

3.1 文件功能的函数接口说明

3.1.1 File_FindFirst

原型	T_U32 File_ FindFirst (T_U16 *path, T_PFINDBUFCTRL bufCtrl)		
功能概述	文件列表搜索功能的初始化，设定缓冲文件的总个数，并填满缓冲；同时获取 path 目录下的目录总个数、文件总个数（非递归）。		
参数说明	path	[in]	要搜索文件夹的绝对路径，unicode 编码
	bufCtrl	[in]	缓冲区控制参数
返回	把搜索控制结构 T_PFINDCTRL 转换成 T_S32 输出，为零则认为是没有找到		
注意事项			

原型	T_U32 File_ FindFirst (T_U16 *path, T_PFINDBUFCTRL bufCtrl)

调用示例	查找 a 盘文件夹 SYSTEM 下面所有的文件
	T_U16 ptr[] = {'A', ':', '/', 'S', 'Y', 'S', 'T', 'E', 'M', '/', '\0'};
	T_U16 pattern[] = {'*', '.', '*', '\0'};
	T_U32 find;
	findBuf.pattern = pattern; //这里是过滤条件
	findBuf.patternLen = (T_U8)Utl_UStrLen(findBuf.pattern);// 过滤条件的长度
	
	findBuf.NodeCnt= 1; //这里是查找缓存个数
	findBuf.type = FILTER_NOTITERATE; //过滤类型，这个是不循环查找
	findBuf.DspCtrl= 0; //不显示".", ".."
	find = File_FindFirst(ptr, &findBuf);
	
	T_U16 Utl_UStrLen(const T_U16* strMain)
	{
	T_U16 len = 0;
	
	if (strMain == 0)
	return 0;
	
	while(*(strMain+len) != 0x00)
	len++;
	return len;
	}

3.1.2 File_FindNext

原型	T_U32 File_ FindNext (T_U32 pFindCtrl, T_S32 Cnt)		
功能概述	从上次搜索位置开始，搜索 Cnt 个文件覆盖至相应的缓冲中		
参数说明	pFindCtrl	[in]	内部为把该参数转换成 T_PFINDCTRL 搜索控制结构体的指针来应用
	Cnt	[in]	搜索文件的个数，只参数只支持-1 与 1,负数时表示向上搜索-个文件；正整数时表示向下搜索一个文件
返回值说明	实际搜索到的文件个数		
注意事项	此函数总是先搜索目录类型的文件，然后再搜索非目录文件。		

调用示例	find = File_FindFirst(ptr, &findBuf);
	File_ FindNext(find, 1); //查找 1 个文件到缓存中
	File_FindClose (T_U32 pFindCtrl)
	详细见 4、File_FindInfo		

3.1.3 File_FindClose

原型	T_VOID File_FindClose (T_U32 pFindCtrl)		
功能概述	关闭接口，释放所有已分配的内存空间		
参数说明	pFindCtrl	[in]	内部为把该参数转换成 T_PFINDCTRL 搜索控制结构体的指针来应用
返回	无		
注意事项	与 File_ FindFirst (T_U16 *path, T_PFINDBUFCTRL bufCtrl)配对使用,否则出现内存泄漏		
调用示例	详细见 4、File_FindInfo		

3.1.4 File_FindFirstFromHandle

原型	T_U32 File_FindFirstFromHandle(T_PFILE file, T_PFINDBUFCTRL bufCtrl)
功能概述	此查找的接口，主要是通过打开文件夹的句柄进行查找，功能上可以不需要在每一次递归查找时进行用 file_fileopen 打开，只需要用 file_findopen 打开，速度上有很大的提升，文件越多，提升越明显。

原型	T_U32 File_FindFirstFromHandle(T_PFILE file, T_PFINDBUFCTRL bufCtrl)		
参数说明	T_PFILE file,	[in]	要搜索文件夹的句柄
	bufCtrl	[in]	缓冲区控制参数
返回	把搜索控制结构 T_PFINDCTRL 转换成 T_S32 输出，为零则认为是没有找到		
注意事项			

调用示例	查找 a 盘文件夹 SYSTEM 下面所有的文件
	T_U8 *file = ak_null;
	T_U16 pattern[] = {'*', '.', '*', '\0'};
	T_U32 find;
	findBuf.pattern = pattern; //这里是过滤条件
	findBuf.patternLen = (T_U8)Utl_UStrLen(findBuf.pattern);// 过滤条件的长度
	findBuf.NodeCnt= 1; //这里是查找缓存个数
	findBuf.type = FILTER_NOTITERATE; //过滤类型，这个是不循环查找
	findBuf.DspCtrl= 0; //不显示".", ".."
	
	file = file_fileAscopen(ak_null, "a:\\system", 0)
	find = File_FindFirstFromHandle(file , &findBuf);
	Do{
	}File_ FindNext(find, 1);
	File_FindCloseWithHandle(T_U32 obj)		

3.1.5 File_FindCloseWithHandle

原型	T_VOID File_FindCloseWithHandle(T_U32 obj)		
功能概述	关闭接口，释放所有已分配的内存空间		
参数说明	pFindCtrl	[in]	内部为把该参数转换成 T_PFINDCTRL 搜索控制结构体的指针来应用
返回值说明	无		
注意事项	File_FindFirstFromHandle(file , &findBuf);配对使用,否则出现内存泄漏		

原型	T_VOID File_FindCloseWithHandle(T_U32 obj)

调用示例	find = File_FindFirstFromHandle(ptr, &findBuf);
	File_ FindNext(find, 1); //查找 1 个文件到缓存中
	File_FindCloseWithHandle(T_U32 obj)

3.1.6 File_FindFirstEX

原　　型	T_U32 File_FindFirstEX_(const T_U16 *path, T_U16 *pattern, T_U16 *fileter_pattern, T_U32 PreFDT);
功能概述	查找文件，此功能与上面的查找功能区别在于，在一个文件夹内查找一个文件后就马上退出来，下一次查找，可以根据 PreFDT 进行从当前位置查找下一个文件
参数说明	const T_U16 *path 传入打开此文件的路径
	T_U16 *pattern 查找配对的类型
	T_U16 *fileter_pattern, 过滤的类型，如果有这个类型就继续查找
	T_U32 PreFDT 上一次查找文件的 fdt
返回值说明	操作成功返回 T_U32 类型 ,否则返回 0
注意事项	与 T_U32 File_FindNextEX(T_U32 obj)配对使用。
调用示例	

3.1.7 File_FindNextEX

原　　型	T_U32 File_FindNextEX(T_U32 obj)
功能概述	查找下一个文件
参数说明	T_U32 obj FileFindFirstEX 返回来的值勤
返回值说明	操作成功返回 T_U32 类型 ,否则返回 0
注意事项	与 T_U32 File_FileFindFirstEX(T_U32 obj)配对使用。

原　　型	T_U32 File_FindNextEX(T_U32 obj)
调用示例	

3.1.8 File_FindFirst_withID

原型	T_U32 File_FindFirst_withID(const T_U16 *path, T_U32 *ret_cnt, T_U16 path_buf[FIND_FILENAME_PATHLEN], T_PFINDBUFINFO find_fileinfo)		
功能概述	对于如下二个接口是进行查找文件使用，查找一个文件就退出来具体支持的功能如下： 1.支持断点查找的功能，比如在查找到一个文件后，退出来查找后．再通过退出前的信息 infobeforeppower，再次进入，查找的是之前退出来的文件． 2.向前向后的查找，向前查找时，如果查找到最后一个文件后，再查找就没有找查找返回查找到的文件个数为 0．而向后查找可以支持循环查找； 3.支持过滤不需要查找的 fileter_pattern，支持需要查找的类型 pattern． 4.支持深层查找（这里的深层查找，只需要在外面传一个 deepflag 值，接口内就可以支持深层查找的功能） 5.支持最大的查找层数是 6 层，超过第 6 层不进行查找. 6.支持获取文件的全路径的功能（path_buf 的内存是内外面分柄传进来，如果传进来的内存为空，那么就不支去获取全路径的功能,要注意的是定义 path_buf 时,path_buf 的空间大小必需是262,，如果小于，那么就会有可能出现内存越界。这个 path_buf 的大小已在头文件 file.h 里定义了 FIND_FILENAME_PATHLEN,所以这个定义如：T_U16 path_buf[FIND_FILENAME_PATHLEN] = {0}, ） 这里所说的向前是 1，向后查找是-1.		
参数说明	const T_U16 *path,	[in]	要搜索文件夹的绝对路径，unicode 编码
	T_U32 *ret_cnt,	[out]	查找返回的个数，查找成功返回 1，没有找到返回 0

原型	T_U32 File_FindFirst_withID(const T_U16 *path, T_U32 *ret_cnt, T_U16 path_buf[FIND_FILENAME_PATHLEN], T_PFINDBUFINFO find_fileinfo)		
	T_U16 path_buf[FIND_FILENAME_PATHLEN],	[in/out]	由平台进行外面分配一个内存，文件系统内存行指向此处，如果需要获取查找的文件全路径，需要传一个空间，如果不需要，那么传 ak_null 就可以了. 注意，定义这个 buf 时，buf 的大小必须是 FIND_FILENAME_PATHLEN，也就是最大的路径 262，如果小于这个值，那么就会出现内存越界的问题。

	T_PFINDBUFINFO find_fileinfo)	in	查找的结构体信息，
			T_PFILEINFO_BEFOREPOWER find_fileinfo; //记录每一层文件的首簇号和 fdt
			T_U16 *pattern; // need find
			T_U16 *fileter_pattern; //no need find, only folder
			T_U16 find_type; // SEARCH_TYPE_ITERATE or SEARCH_TYPE_NOTITERATE
			T_BOOL deep_flag; //deep find or not

返回	把搜索控制结构 T_PFINDCTRL_PDFILEINFO 转换成 T_U32 的句柄输出， 如果句柄为零和参数 ret_cnt = 0，则认为是没有找到； 如果如果句柄为零和参数 ret_cnt=2，则认为是查找出错了。 而句柄是有值，并参数 ret_cnt=1，那么表示正确查找到一个文件。		
注意事项	path_buf 的内存是内外面分柄传进来，如果传进来的内存为空，那么就不支去获取全路径的功能,要注意的是定义 path_buf 时,path_buf 的空间大小必需是 262,，如果小于 262，那么就会有可能出现内存越界。这个 path_buf 的大小已在头文件 file.h 里定义了 FIND_FILENAME_PATHLEN,所以这个定义如：T_U16 path_buf[FIND_FILENAME_PATHLEN] = {0}, ）		

原型	T_U32 File_FindFirst_withID(const T_U16 *path, T_U32 *ret_cnt, T_U16 path_buf[FIND_FILENAME_PATHLEN], T_PFINDBUFINFO find_fileinfo)

调用示例	查找 a 盘文件夹 SYSTEM 下面所有的文件
	T_U16 path[] = {'A', ':', '/', 'S', 'Y', 'S', 'T', 'E', 'M', '/', '\0'};
	T_U32 find;
	T_U32 ret_cnt = 0;
	T_U16 path_buf[FIND_FILENAME_PATHLEN] = {0};
	T_FINDBUFINFO find_fileinfo;
	T_PFILEINFO_BEFOREPOWER fileinfo;
	
	find_fileinfo.pattern = {'*', '.', '*', '\0'};
	find_fileinfo.fileter_pattern = {'d', 'i', 'r', '\0'};
	find_fileinfo.deep_flag = deepflag;
	find_fileinfo.find_type = find_type;
	find_fileinfo->find_fileinfo = ak_null 或读取保存上一次退出来的查找信息
	
	find = File_FindFirst_withID(path, &ret_cnt, path_buf, &find_fileinfo)
	If(find != 0)
	{
	File_FindInfo_WithID(find, 0, NULL,NULL);
	File_get_findfile_info_WithID(find, fileinfo);
	
	File_FindNext_withID(find, 1, &ret_cnt)
	}while(ret_cnt != 0)
	File_FindCloseWithID(T_U32 obj);

3.1.9 File_FindNext_withID

原型	T_U32 File_FindNext_withID(T_U32 obj, T_S32 Cnt, T_U32 *ret_cnt)		
功能概述	从上次搜索位置开始，搜索 Cnt 个文件覆盖至相应的缓冲中		
参数说明	T_U32 obj	[in]	内部为把该参数转换成 T_PFINDCTRL 搜索控制结构体的指针来应用
	T_S32 Cnt	[in]	向前（1）或向后（-1）查找
	T_U32 *ret_cnt	[out]	查找返回的个数，查找成功返回 1，没有找到返回 0
返回值说明	把搜索控制结构 T_PFINDCTRL_PDFILEINFO 转换成 T_U32 输出， 如果句柄为零和参数 ret_cnt = 0，则认为是没有找到； 如果如果句柄为零和参数 ret_cnt=2，则认为是查找出错了。 而句柄是有值，并参数 ret_cnt=1，那么表示正确查找到一个文件。		
注意事项	此查找的功能只能是一个文件一个文件进行查找		
调用示例	详细见 3.1.8File_FindFirst_withID		

3.1.10 File_FindInfo_WithID

原型	T_PFILEINFO File_FindInfo_WithID(T_U32 obj, T_U32 Position, T_U32 *FileCnt, T_U32 *FolderCnt);		
功能概述	从输入的搜索控制结构体对象 pFindCtrl 中得到文件的信息		
参数说明	T_U32 obj	[in]	输入的搜索控制结构体对象
	T_U32 Position,	[in]	对象中的文件链表的位置
	FileCnt	[out]	符合搜索条件的文件总个数
	FolderCnt	[out]	符合搜索条件的文件夹总个数
返回	查找到的文件的信息 fileInfo		
注意事项			
调用示例	详细见 3.1.8File_FindFirst_withID		

3.1.11 File_get_findfile_info_WithID

原型	T_VOID File_get_findfile_info_WithID(T_U32 obj, T_PFILEINFO_BEFOREPOWER fileinfo);		
功能概述	从输入的搜索控制结构体对象 pFindCtrl 中得到文件的信息		
参数说明	T_U32 obj,	[in]	查找的句柄
	T_PFILEINFO_BEFOREPOWER fileinfo	[out]	查找到文件首簇号和 fdt 的信息
返回	无		
注意事项			
调用示例	详细见 3.1.8File_FindFirst_withID		

3.1.12 File_FindCloseWithID

原型	T_VOID File_FindCloseWithID(T_U32 obj);		
功能概述	关闭接口，释放所有已分配的内存空间		
参数说明	T_U32 obj	[in]	内部为把该参数转换成 T_PFINDCTRL_PDFILEINFO 搜索控制结构体的指针来应用
返回	无		
注意事项	与 File_FindFirst_withID 配对使用,否则出现内存泄漏		
调用示例	详细见 3.1.8File_FindFirst_withID		

3.1.13 File_GetFileinfo_fastopen

原型	T_BOOL File_GetFileinfo_fastopen(T_U32 file, T_PFILEINFO_FASTOPEN fileInfo);		
功能概述	通过文件名柄获取此文件的 fdt，首簇号，父目录首簇号，所在的 driver。		
参数说明	T_U32 file	[in]	文件句柄
	T_PFILEINFO_FASTOPEN fileInfo	[out]	此文件的信息 fdt，首簇号，父目录首簇号等
返回	成功与否		

原型	T_BOOL File_GetFileinfo_fastopen(T_U32 file, T_PFILEINFO_FASTOPEN fileInfo);
注意事项	无
调用示例	

3.1.14 File_fastopen_onlyfile

原型	T_U32 File_fastopen_onlyfile(T_U8 driverID, T_U32 fileID, T_U32 ParentID , T_U32 FDTOffset, T_U32 mode);		
功能概述	通过 File_GetFileinfo_fastopen(T_U32 file, T_PFILEINFO_FASTOPEN fileInfo) 接口获取参数，进行快速打开文件		
参数说明	T_U8 driverID	[in]	Driver 的 id
	T_U32 fileID,	[in]	文件的首簇号
	T_U32 ParentID	[in]	父目录首簇号
	T_U32 FDTOffset	[in]	fdt
	T_U32 mode	[in]	打开文件的模式，只支持只读打开和覆盖打开
返回	返回有效句柄或 0		
注意事项	与 File_close 配对使用,否则出现内存泄漏		
调用示例	以创建方式打开一个文件，获取相应的名 T_U8* FileName = "test.txt" T_PFILEINFO_FASTOPEN fileInfo; T_U32 file = File_OpenAsc(AK_NULL, FileName, FILE_MODE_CREATE); File_flush(file); File_GetFileinfo_fastopen(file, &fileInfo); File_close(file); file = File_fastopen_onlyfile(driverID, fileInfo.fileID, fileInfo.ParentID , fileInfo.FDTOffset, 0); File_close(file);		

3.1.15 File_chkdsk_withfolder

原型	T_BOOL File_chkdsk_withfolder(const T_U8 *path);		
功能概述	根据传进来的路径，进行对此路径下的文件的簇链进行 chkdsk，修改簇链		
参数说明	const T_U8 *path	[in]	传进来的路径
返回	无		
注意事项			
调用示例	File_chkdsk_withfolder(“a:\\test_dir”);		

3.1.16 File_FindInfo

原型	T_PFILEINFO File_FindInfo(T_U32 pFindCtrl, T_U32 Position, T_U32 *FileCnt, T_U32 *FolderCnt)		
功能概述	从输入的搜索控制结构体对象 pFindCtrl 中得到文件的信息		
参数说明	pFindCtrl	[in]	输入的搜索控制结构体对象
	Position	[in]	对象中的文件链表的位置
	FileCnt	[out]	符合搜索条件的文件总个数
	FolderCnt	[out]	符合搜索条件的文件夹总个数
返回	如果文件存在和输出的 fileInfo 不为空返回 AK_TRUE，否则返回 AK_FALSE		
注意事项	必须也 File_FindFirst 配合使用, Position 是找出缓冲个数偏移个数，比如在 findBuf.NodeCnt= 3，那么这个搜索缓存 3 个文件，File_ FindNext(find, 3)也就是可以一次搜索出 3 个文件，那么 Position 为 0 就是找到缓存中的第一个文件，为 1 就是找到缓存中的第二个文件，依次类推		

调用示例	T_PFILEINFO info;
	find = File_FindFirst(ptr, &findBuf);
	do{
	info = File_FindInfo(find, 0, AK_NULL, AK_NULL);
	
	}while(1 == File_ FindNext(find, 1));
	File_FindClose (find);		

3.1.17 File_OpenAsc

原型	T_PFILE File_OpenAsc(T_PFILE parent, const T_U8* pathname, T_U32 mode)		
功能概述	以 ascii 码方式打开文件		
参数说明	parent	[in]	当前的文件夹的文件结构体指针，如果等于 AK_NULL 就认为是从根目录开始
	pathname	[in]	要打开的文件路径，ascii 编码
	mode	[in]	打开的方式： 1、FILE_MODE_READ 只读方式打开无论文件或者目录是否存在多返回一个文件对象,此方式不管文件是否存在,都会返回一个文件句柄. 2、FILE_MODE_CREATE 创建的方式打开，如果打开的文件存在，先删除，后创建 3、FILE_MODE_OVERLAY 覆盖方式打开，如果文件存在，返回文件指针的头部 4、FILE_MODE_APPEND 追加方式打开，如果文件存在，返回文件指针的尾部 5、FILE_MODE_EXT_NO_SEPARATE_FAT 和 2、3、4 点与后配合应用，主要是是否应用分次写 FAT 表的问题了，该修改点会影响单次写耗时，如果不分次写，那么分次开销将在文件关闭的时候开销
返回	指向文件结构体的指针，输入参数错误，或者删除，读数据失败返回空指针		
注意事项	无论文件是否存在，都返回一个正确的文件结构体，需要通过判断 File_Exist 来判断文件是否存在。当初为了创建文件夹而设计的。 另外对于同一文件多次使用 OPEN 打开的话,系统只允许多次只读打开,如果先使用非只读方式打开,之后再次打开将返回失败,如果首次以只读方式打开的话,将会只允许多次以只读方式打开, 否则将返回失败.		

原型	T_PFILE File_OpenAsc(T_PFILE parent, const T_U8* pathname, T_U32 mode)
调用示例	**新增的应用示例：** 应用方法：如果不需要，和之前一样应用好了，如果需要那么就在创建方式的时候增加一个宏控制 以 11 为例：11 的 sec 都是 512 如果少于 4G，簇都是 8 个扇区，所以一个簇就是 4k 大小，如果簇比 page 小，那么簇就是 page 的大小，以 8kpage 的 nand 为例，簇就是 8k，FAT32 文件系统计算，那么刷 fat 表的阀值就是 8/4=2k 个簇，2k*8k=6M 的文件就刷新一次 fat T_PFILE File_OpenAsc(T_PFILE parent, const T_U8* FileName, T_U32 mode); 参数 parent 可以是 AK_NULL 或者打开的文件夹句柄 参数 FileName 是 ASCII 的需要打开文件路径 mode = FILE_MODE_CREATE | FILE_MODE_EXT_NO_SEPARATE_FAT **这个方式就是不分次刷新 FAT 表**

3.1.18 File_OpenUnicode

原型	T_PFILE File_OpenUnicode(T_PFILE parent, const T_U16* pathname, T_U32 mode)		
功能概述	以 Unicode 码方式打开文件		
参数说明	parent	[in]	当前的文件夹的文件结构体指针，如果等于 AK_NULL 就认为是从根目录开始
	pathname	[in]	要打开的文件路径，unicode 编码
	mode	[in]	打开的方式，参考 File_OpenAsc
返回	指向文件结构体的指针，输入参数错误，或者删除，读数据失败返回空指针		
注意事项	无论文件是否存在，都返回一个正确的文件结构体，，需要通过判断 File_Exist 来判断文件是否存在。当初为了创建文件夹而设计的。		
调用示例			

3.1.19 File_MkdirsUnicode

原型	T_BOOL File_MkdirsUnicode(const T_U16* path)		
功能概述	创建多个目录下的文件夹		
参数说明	path	[in]	输入的是 unicode 码的路径
返回	创建成功返回 AK_TRUE，失败时返回 AK_FALSE		

原型	T_BOOL File_MkdirsUnicode(const T_U16* path)
注意事项	
调用示例	

3.1.20 File_MkdirsAsc

原型	T_BOOL File_MkdirsAsc(const T_U8* path)		
功能概述	创建多个目录下的文件夹		
参数说明	Path	[in]	输入的是 ascii 码的路径
返回	创建成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项			
调用示例			

3.1.21 File_Close

原型	T_VOID File_Close(T_PFILE file)		
功能概述	文件的关闭		
参数说明	file	[in]	文件夹的文件结构指针
返回	无		
注意事项	调用此函数后,将不能再使用此文件指针对文件进行操作,否则将会有不可预期的情况,因为调用后,文件指针的内存已经被释放,是个野指针.		
调用示例			

3.1.22 File_DelFile

原型	T_BOOL File_DelFile(T_PFILE file)		
功能概述	文件删除		
参数说明	file	[in]	文件夹的文件结构指针
返回	创建成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项	这个主要针对已经打开的文件删除。删除文件夹用 File_DelDir		

原型	T_BOOL File_DelFile(T_PFILE file)
调用示例	**T_PFILE file = AK_NULL;** **file = File_OpenUnicode(AK_NULL, FileName, FILE_MODE_READ);** **File_DelFile(file);** **File_Close(file);**

3.1.23 File_DelDir

原型	T_BOOL File_DelDir(T_PFILE file)		
功能概述	删除整个文件夹目录下的子目录和子文件		
参数说明	file	[in]	文件夹的文件结构体
返回	删除成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项	比如删除 test, test 中有 3 个文件，那么返回值就是 3+1=4		

调用示例	T_PFILE file = AK_NULL;
	file = OpenAsc(AK_NULL, "C:\\test", **FILE_MODE_READ**);
	File_DelDir(file); //在内部有判断是否文件夹的
	File_Close(file);		

3.1.24 File_DelAsc

原型	T_BOOL File_DelAsc(const T_U8 *FileName)		
功能概述	文件或者文件夹的删除		
参数说明	FileName	[in]	输入的是 ascii 码的路径
返回	删除成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项			
调用示例			

3.1.25 File_DelUnicode

原型	T_BOOL File_DelUnicode(const T_U16 *FileName)		
功能概述	文件或者文件夹的删除		
参数说明	FileName	[in]	输入的是 unicode 码的路径
返回	删除成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项			
调用示例			

3.1.26 File_DelAsc_DelCallBack

原型	T_BOOL File_DelAsc_DelCallBack(const T_U8 *FileName, F_DelCallback delCallBack, T_VOID *delCallBackData)		
功能概述	文件或者文件夹的删除，删除的过程中，可以通过 delCallBack 回调函数进行退出删除。		
参数说明	FileName	[in]	输入的是 unicode 码的路径
	F_DelCallback delCallBack	[in]	回调函数
	T_VOID *delCallBackData		暂无用
参数说明	FileName	[in]	输入的是 ascii 码的路径
返回	删除成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项			
调用示例	**typedef T_BOOL F_DelCallback(T_VOID *pData, T_U16 *FileName);**		

3.1.27 File_DelUnicode_DelCallBack

原型	T_BOOL File_DelUnicode_DelCallBack (const T_U16 *FileName, F_DelCallback delCallBack, T_VOID *delCallBackData)		
功能概述	文件或者文件夹的删除，删除的过程中，可以通过 delCallBack 回调函数进行退出删除。		
参数说明	FileName	[in]	输入的是 unicode 码的路径

原型	T_BOOL File_DelUnicode_DelCallBack (const T_U16 *FileName, F_DelCallback delCallBack, T_VOID *delCallBackData)		
	F_DelCallback delCallBack	[in]	回调函数
	T_VOID *delCallBackData		暂无用
返回	删除成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项			
调用示例	**typedef T_BOOL F_DelCallback(T_VOID *pData, T_U16 *FileName);**		

3.1.28 File_Exist

原型	T_BOOL File_Exist(T_PFILE file);		
功能概述	检测文件是否存在		
参数说明	file	[in]	文件夹的文件结构指针
返回	存在返回 AK_TRUE，不存在时返回 AK_FALSE		
注意事项			
调用示例			

3.1.29 File_IsFolder

原型	T_BOOL File_IsFolder(T_PFILE file)		
功能概述	判断文件结构体指针是否文件夹		
参数说明	file	[in]	文件夹的文件结构指针
返回	是返回 AK_TRUE，不是时返回 AK_FALSE		
注意事项			
调用示例			

3.1.30 File_IsFile

原型	T_BOOL File_IsFile(T_PFILE file)		
功能概述	判断文件结构体指针是否文件		
参数说明	file	[in]	文件夹的文件结构指针
返回	是返回 AK_TRUE，不是时返回 AK_FALSE		
注意事项			
调用示例			

3.1.31 File_SetBufferSize

原型	T_BOOL File_SetBufferSize(T_PFILE file, T_U32 SectorNum)		
功能概述	设置打开文件的 databuffer 大小，合理的调整可以提高写入性能		
参数说明	file	[in]	已经打开的文件的文件结构体指针,
	SectorNum	[in]	设置的文件 databuffer 大小为 SectorNum 个扇区大小
返回	设置成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项	如果是文件夹指针,将直接返回失败		
调用示例			

3.1.32 File_GetModTime

原型	T_BOOL File_GetModTime(T_PFILE file, T_PFILETIME fileTime)		
功能概述	得到文件的修改时间		
参数说明	file	[in]	已经打开的文件的文件结构体指针
	fileTime	[out]	输出的文件时间的机构体指针
返回	成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项			
调用示例			

3.1.33 File_GetCreateTime

原型	T_BOOL File_GetCreateTime(T_PFILE file, T_PFILETIME fileCTime)		
功能概述	得到文件的创建时间		
参数说明	file	[in]	已经打开的文件的文件结构体指针
	fileTime	[out]	输出的文件时间的机构体指针
返回	成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项			
调用示例			

3.1.34 File_RenameTo

原型	T_BOOL File_RenameTo(T_PFILE source, T_PFILE dest)		
功能概述	文件重命名或者搬移文件		
参数说明	T_PFILE source	[in]	已经打开的源文件的文件结构体指针
	T_PFILE dest	[in]	已经打开的目的文件的文件结构体指针
返回	操作成功返回 AK_TRUE，失败时返回 AK_FALSE		
注意事项	**不支持不同盘符的文件之间的命名与移动**		
调用示例			

3.1.35 File_CopyAsc

原型	T_BOOL File_CopyFileAsc(const T_U8* srcPath, const T_U8* dstPath, T_BOOL replace， F_CopyCallback pCallBack, T_VOID *pCallBackData)		
功能概述	拷贝 sourFile 文件(或文件夹中的文件)到 destFile 文件(或文件夹)中，拷贝的过程中，可以通过 delCallBack 回调函数进行退出拷贝或获取拷贝文件的情况。		
参数说明	srcPath	[in]	源文件路径(ASCII 格式)
	dstPath	[in]	目的文件路径(ASCII 格式)
	replace	[in]	如果目标文件存在,是否替换
	F_CopyCallback pCallBack	[in]	回调函数
	T_VOID *pCallBackData		无用

原型	T_BOOL File_CopyFileAsc(const T_U8* srcPath, const T_U8* dstPath, T_BOOL replace， F_CopyCallback pCallBack, T_VOID *pCallBackData)
返回	操作成功返回 AK_TRUE, 否则返回 AK_NULL
注意事项	不能文件夹拷贝到文件， 如果是文件拷贝到文件，则覆盖(源文件的长度)目的文件的内容，如果目的文件夹中有相同的文件名，则完全取代。
调用示例	typedef T_BOOL F_CopyCallback(T_VOID *pData, T_U16 *FileName,T_U32 CurPos, T_U32 FileSize);

3.1.36 File_CopyUnicode

原型	T_BOOL File_CopyFileUnicode(const T_U16* srcPath, const T_U16* dstPath, T_BOOL replace， F_CopyCallback pCallBack, T_VOID *pCallBackData)		
功能概述	拷贝 sourFile 文件(者文件夹中的文件)到 destFile 文件(者文件夹)中，拷贝的过程中，可以通过 delCallBack 回调函数进行退出拷贝或获取拷贝文件的情况		
参数说明	srcPath	[in]	源文件路径(UNICODE 格式)
	dstPath	[in]	目的文件路径(UNICODE 格式)
	replace	[in]	如果目标文件存在,是否替换
	F_CopyCallback pCallBack	[in]	回调函数
	T_VOID *pCallBackData		无用
返回	操作成功返回 AK_TRUE, 否则返回 AK_NULL		
注意事项	不能文件夹拷贝到文件， 如果是文件拷贝到文件，则覆盖(源文件的长度)目的文件的内容，如果目的文件夹中有相同的文件名，则完全取代。		
调用示例	typedef T_BOOL F_CopyCallback(T_VOID *pData, T_U16 *FileName,T_U32 CurPos, T_U32 FileSize);		

3.1.37 File_Read

原型	T_U32 File_Read (T_PFILE file, T_pVOID buf, T_U32 byts)		
功能概述	文件读取数据		
参数说明	file	[in]	需要读取的文件结构体指针
	buf	[in]	读取存放的 buf

原型	T_U32 File_Read (T_PFILE file, T_pVOID buf, T_U32 byts)		
	byts	[in]	需要读取的字节数
返回	成功读取的字节数		
注意事项	如果返回和要读出的字节 byts 不等，要么读到文件的结尾，要么是出错了 可以通过读取 File_GetFilePtr 和 File_GetLength 来之得知是否到了结尾, 可以通过判断 file->ValidFlag == AK_TRUE 知道数据是否有效, 另外文件夹指针将直接返回失败.		
调用示例			

3.1.38 File_Write

原型	T_U32 File_Write(T_PFILE file, T_pVOID buf, T_U32 byts)		
功能概述	文件写入数据		
参数说明	file	[in]	需要写入的文件结构体指针
	buf	[in]	需要写入的数据 buf
	byts	[in]	需要写入的字节数
返回	成功写入的字节数		
注意事项	如果返回和要写入的字节 byts 不等，要么没有空间了， 要么是出错了 可以通过判断 file->ValidFlag == AK_TRUE 知道数据是否有效, 另外文件夹指针将直接返回失败.		
调用示例			

3.1.39 File_GetOccupiedSpace

原型	T_U32 File_GetOccupiedSpace(T_PFILE file)		
功能概述	得到文件实际占用的空间		
参数说明	file	[in]	文件夹的文件结构指针
返回	返回文件实际占用的空间		
注意事项	这个和文件大小有点区别，因为存储文件的时候都是以簇为单位的，所以得到的实际战役空间一定是簇的整数倍		

原型	T_U32 File_GetOccupiedSpace(T_PFILE file)
调用示例	

3.1.40 File_Flush

原型	T_VOID File_Flush(T_PFILE file)		
功能概述	刷新文件的数据		
参数说明	file	[in]	需要刷新的文件的文件结构指针
返回	无		
注意事项			
调用示例			

3.1.41 File_SetDefault

原型	T_BOOL File_SetDefault(T_PFILE file)		
功能概述	如果 file 是文件夹，那么设置该文件夹为默认的路径		
参数说明	file	[in]	文件夹的文件结构指针
返回	设置成功返回 AK_TRUE，失败 AK_FALSE		
注意事项			
调用示例			

3.1.42 File_GetAttrAsc

原型	T_U32 File_GetAttrAsc(const T_U8 *path)		
功能概述	以输入 ascii 路径方式得到文件的属性		
参数说明	path	[in]	输入 ascii 路径
返回	如果存在则文件的属性， 如果不存在返回 0xffffffff		
注意事项	只有低四位有效,文件的属性.		
调用示例			

3.1.43 File_GetAttrUnicode

原型	T_U32 File_GetAttrUnicode (const T_U16 *path)		
功能概述	以输入 unicode 路径方式得到文件的属性		
参数说明	path	[in]	输入 unicode 路径
返回	如果存在则文件的属性， 如果不存在返回 0xffffffff		
注意事项	只有低四位有效,文件的属性.		
调用示例			

3.1.44 File_SetAttrAsc

原型	T_U32 File_SetAttrAsc(const T_U8 *path, T_U32 attribute)		
功能概述	以输入 ascii 路径方式设置文件的属性		
参数说明	path	[in]	输入 ascii 路径
	attribute	[in]	需要设置的属性
返回	如果存在则文件的属性， 如果不存在返回 0xffffffff		
注意事项	只有低四位有效,文件的属性.如果是文件夹的话,将不能修改文件夹内的文件属性		
调用示例			

3.1.45 File_SetAttrUnicode

原型	T_U32 File_SetAttrUnicode (const T_U16 *path, T_U32 attribute)		
功能概述	以输入 unicode 路径方式设置文件的属性		
参数说明	path	[in]	输入 unicode 路径
	attribute	[in]	需要设置的属性
返回	如果存在则文件的属性， 如果不存在返回 0xffffffff		
注意事项	只有低四位有效,文件的属性.如果是文件夹的话,将不能修改文件夹内的文件属性		
调用示例			

3.1.46 File_GetFileinfo

原型	T_BOOL File_GetFileinfo(T_PFILE file, T_PFILEINFO fileInfo)		
功能概述	从输入的文件结构体中得到文件的信息		
参数说明	file	[in]	输入的文件结构体
	fileInfo	[out]	输出的文件信息
返回	如果文件存在和输出的 fileInfo 不为空返回 AK_TRUE，否则返回 AK_FALSE		
注意事项	可能通过 fileInfo 这个结构体获取文件的相应的 fdt 值		
调用示例			

3.1.47 File_GetLength

原型	T_U32 File_GetLength(T_PFILE file, T_U32* high)		
功能概述	从输入的文件结构体中得到文件的长度大小		
参数说明	file	[in]	输入的文件结构体
	high	[out]	输出的文件长度的高 32 位
返回	输出的文件长度的低 32 位		
注意事项			
调用示例			

3.1.48 File_GetFilePtr

原型	T_U32 File_GetFilePtr(T_PFILE file, T_U32 *high)		
功能概述	得到文件当前的指针		
参数说明	file	[in]	输入的文件结构体
	high	[out]	文件指针的高 32 位
返回	如果文件存在，返回文件当前的指针的低 32 位，否则返回 0xffffffff		
注意事项	该函数和 windows 的 ftell 相当		
调用示例			

3.1.49 File_SetFilePtr

原型	T_U32 File_SetFilePtr (T_PFILE file, T_S32 offset, T_U16 origin)		
功能概述	定位当前打开的文件的指针		
参数说明	file	[in]	打开文件的文件结构指针
	offset	[in]	需要定位的方式的偏移量，负数为向前偏移
	origin	[in]	定位的方式： FILE_SEEK_SET 开始位置偏移 FILE_SEEK_CUR 当前位置偏移 FILE_SEEK_END 尾巴位置偏移
返回	文件不存在或者没有打开时返回 0xffffffff，如果搜索长度大于文件长度截取到文件结尾，如果向后搜索长度小于开始， 截取到开始。文件读写(如果有新的数据没有写入需要写操作)失败，直接返回当前文件的指针，否则返回所需要搜索的指针		
注意事项	注意事项，由于文件最大取值为 T_U32(4g)，而设置偏移最大为 T_S32(-2g~2g)，所以如果设置偏移超过 2g，比如象定位到当前指针偏移 3g： T_U32 ptr = 3*1024*0124; File_SetFilePtr(file, 2*1024*1024, FILE_SEEK_CUR); File_SetFilePtr(file, 1*1024*1024, FILE_S_CUR); 那么用两次 File_SetFilePtr 即可		
调用示例			

3.1.50 File_Truncate

原型	T_BOOL File_Truncate(T_PFILE file, T_U32 low, T_U32 high)		
功能概述	从输入的文件结构体中截取 从文件开始 length 长度的文件		
参数说明	file	[in]	输入的文件结构体
	low	[in]	需要截取的文件的长度的低 32 位
	length	[in]	需要截取的文件的长度的高 32 位
返回	成功返回 AK_TRUE, 失败返回 AK_FALSE，也就是截取不成功		

原型	T_BOOL File_Truncate(T_PFILE file, T_U32 low, T_U32 high)
注意事项	假如输入的长度大于文件的长度，返回失败。截取后需要调用 File_Close 来更新文件，不能直接先调用 File_Destroy,文件夹将直接返回失败
调用示例	

3.1.51 File_GetPathObj

原型	T_U32 File_GetPathObj(T_PFILE file)		
功能概述	得到文件全路径的指针对象		
参数说明	File	[in]	输入的文件结构体
返回	如果文件存在，返回文件全路径的指针对象，否则返回 0		
注意事项	一定要和 File_GetAbsPath 、File_DestoryPathObj 配对使用，如果不调用会有 File_DestoryPathObj 内存泄露		
调用示例			

3.1.52 File_GetAbsPath

原型	T_U16* File_GetAbsPath(T_U32 obj)		
功能概述	得到文件的全路径(注意，这里是 unicode 码输出)		
参数说明	obj	[in]	输入的文件全路径的指针对象
返回	如果文件存在，返回文件文件的全路径指针地址，否则返回 AK_NULL		
注意事项	一定要和 File_GetAbsPath 、File_DestoryPathObj 配对使用		
调用示例			

3.1.53 File_DestroyPathObj

原型	T_VOID File_DestroyPathObj(T_U32 obj)		
功能概述	销毁文件全路径的指针对象		
参数说明	obj	[in]	输入的文件全路径的指针对象
返回	无		
注意事项	一定要和 File_GetAbsPath 、File_DestoryPathObj 配对使用		

原型	T_VOID File_DestroyPathObj(T_U32 obj)
调用示例	

3.1.54 File_SetFileSize

原　　型	T_BOOL File_SetFileSize(T_PFILE file, T_U32 fileSize);
功能概述	设置创建文件的大小,此功能是文件系统中新增加功能,使用此功能,如果写文件将会先分配此大小的簇给文件使用,使用完之后将会再分配同样大小的空间,以便保证文件的连续性。
参数说明	T_PFILE file:文件指针
	T_U32 fileSize:预设文件大小
返回值说明	操作成功返回 1,否则返回 0.
注意事项	这个参数要根据实际使用合理安排,文件夹无此功能
调用示例	File_SetFileSize(pFile,100*1024*1024);//分配 100M

3.1.55 File_CreateVolumeAsc

原　　型	T_BOOL File_CreateVolumeAsc(const T_U8* FileName)
功能概述	以 ASCII 方式创建磁盘的卷标文件
参数说明	FileName 卷标的名字(ASCII)，通过盘符指定创建哪个盘符的卷标
返回值说明	操作成功返回 1,否则返回 0.
注意事项	卷标名字最大就 11 个字节，如果输入的带盘符的是 13 个字节，不带盘符的输入就是默认创建文件系统内部默认的盘符路径的卷标，初始化是 A 盘
调用示例	File_CreateVolumeAsc ("B:\\anyka");//创建 B 盘的卷标名字是"anyka"

3.1.56 File_CreateVolumeUnicode

原　　型	T_BOOL File_CreateVolumeUnicode(const T_U16* FileName)
功能概述	以 UNICODE 方式创建磁盘的卷标文件
参数说明	FileName 卷标的名字(UNICODE)，通过盘符指定创建哪个盘符的卷标

原　　型	T_BOOL File_CreateVolumeUnicode(const T_U16* FileName)
返回值说明	操作成功返回 1,否则返回 0.
注意事项	卷标名字最大就 11 个字节，如果输入的带盘符的是 13 个字节，不带盘符的输入就是默认创建文件系统内部默认的盘符路径的卷标，初始化是 A 盘
调用示例	雷同 T_BOOL File_CreateVolumeAsc(const T_U8* FileName)

3.2 Driver层函数接口定义

3.2.1 Driver_Format

原型	T_BOOL Driver_Format(T_U8 DriverID, T_U32 BufLen, E_FATFS fsType)		
功能概述	Driver 的格式化,此接口是可以格式化成指定的文件系统类型。		
参数说明	DriverID	[in]	分区的 ID
	BufLen		输入 medium 的 fdt 和 fat 缓冲区的长度(字节数)，越大，目录缓冲能力越强。
	fsType		文件系统的类型,目前只支持 FAT,EXFAT typedef enum { FAT_FS_ERROR = 0, FAT_FS_16 = 1, FAT_FS_32 = 2, FAT_FS_EX = 3 }E_FATFS;
返回	如果成功返回 TRUE,失败返回 FALSE		
注意事项	对于 BufLen 的实现 BufSecNum = BufLen >> medium->SecBit; 其中 BufSecNum 根据下面来获取缓冲能力 [0, 5), [5, 9), [9, 13), [13, ~~) Fat 1 2 3 BufSecNum >> 2; Fdt = BufSecNum – Fat; 如果 BufSecNum=0, 1 那么 fdt = 1		

原型	T_BOOL Driver_Format(T_U8 DriverID, T_U32 BufLen, E_FATFS fsType)		
调用示例	Driver_Format (0, PageSize, FAT_FS_EX);		

3.2.2 Driver_QuickFormat

原型	T_BOOL Driver_QuickFormat(T_U8 DriverID)		
功能概述	此接口也 Driver 的格式化，但这个接口只是对现有的分区进行快速格式化而已，保留原有的文件系统类型。		
参数说明	DriverID	[in]	分区的 ID
返回	**如果成功返回 TRUE,失败返回 FALSE**		
注意事项			
调用示例	Driver_QuickFormat(0);		

3.2.3 Driver_GetCapacity

原型	T_U32 Driver_GetCapacity(T_U8 driverID, T_U32 *high)		
功能概述	得到 Driver 的总容量 单位是字节		
参数说明	driverID	[in]	对应分区的 ID
	T_U32 *high	[out]	获得的总容量的高 32 位
返回	获得的总容量的低 32 位		
注意事项			
调用示例			

3.2.4 Driver_GetUsedSize

原型	T_U32 Driver_GetUsedSize(T_U8 driverID, T_U32 *high)		
功能概述	得到 Driver 的已用空间 单位是字节		
参数说明	driverID	[in]	对应分区的 ID

原型	T_U32 Driver_GetUsedSize(T_U8 driverID, T_U32 *high)		
	high	[out]	获得的已用空间的高 32 位
返回	获得的已用空间的低 32 位		
注意事项			
调用示例			

3.2.5 Driver_GetFreeSize

原型	T_U32 Driver_GetFreeSize (T_U8 diverID, T_U32 *high)		
功能概述	得到 Driver 的剩余空间 单位是字节		
参数说明	driverID	[in]	对应分区的 ID
	high	[out]	获得的剩余空间的高 32 位
返回	获得的剩余空间的低 32 位		
注意事项			
调用示例			

3.2.6 Driver_ClearBuf

原型	T_VOID Driver_ClearBuf(T_U8 DriverId)		
功能概述	清空相应 Drive 的 fdt 和 fat 的 buffer		
参数说明	DriverId	[in]	需要清空的 driver 的 ID
返回	无		
注意事项	在读写文件之前调用这个接口有可能会提高写入速度，例如：假如缓存已经很多或者满了，就有可以因为搜索缓存时间长或者还要申请新的缓存影响速度，不过假如读写的文件的 fdt 和 fat 刚好在缓存中，如果清空由于还要从媒介中读取，那就反而变慢。应用当中不调用这个函数也不会有问题。		
调用示例			

3.2.7 Driver_GetDriverIDAsc

原型	T_U8 Driver_GetDriverIDAsc (const T_U8 *path)		
功能概述	从输入的路径中获得 Driver 的 ID		
参数说明	path	[in]	输入的路径，编码是 ascii 码
返回	Driver 的 ID，如三个盘分别可以为 0，1，2,如果没有将返回-1		
注意事项			
调用示例			

3.2.8 Driver_GetDriverIDUnicode

原型	T_U8 Driver_GetDriverIDUnicode (const T_U16 *path)		
功能概述	从输入的路径中获得 Driver 的 ID		
参数说明	path	[in]	输入的路径，编码是 unicode 码
返回	Driver 的 ID，如三个盘分别可以为 0，1，2,如果没有将返回-1		
注意事项			
调用示例			

3.3 全局函数接口定义

3.3.1 FS_InitCallBack

原　　型	T_BOOL FS_InitCallBack(T_PFSCallback fsInitInfo, T_U16 PagesPerFSBuf)
功能概述	初始化文件系统中可能会使用到的回调函数
参数说明	T_PFSCallback fsInitInfo :文件系统接口回调函数
	T_U16 PagesPerFSBuf: 文件系统层缓冲区大小,是以页为单位.
返回值说明	操作成功返回 1,否则返回 0.
注意事项	在系统使用文件系统相关功能前,必须先调用此函数.
调用示例	FS_InitCallBack (fsInitInfo, 128);

3.3.2 FS_MountNandFlash

原　　型	T_U8 FS_MountNandFlash(T_PNANDFLASH gNand, T_U8 *DriverCnt)
功能概述	加载 NANDFLASH,统计 NANDFLASH 的分区信息.
参数说明	T_PNANDFLASH gNand: NAND 的结构体信息.
	T_U8 *DriverCnt: NANDFLASH 中总共有多少个分区.
返回值说明	返回系统为此 NANDFLASH 分配的第一个分区的 ID.
注意事项	在系统使用 NANDFLASH 相关功能前,必须先调用此函数.
调用示例	StartDriver = FS_MountNandFlash(gNand, &DriverCnt);

3.3.3 FS_MountMemDev

原　　型	T_U8 FS_MountMemDev(PDRIVER _INFO pDevInfo, T_U8 *DriverCnt);
功能概述	当系统启动时,调用此函数后才能使用相应移动储存设备的文件系统功能与 U 盘功能.
参数说明	PDRIVER_INFO pDevInfo: 相应移动储存设备相关的读写与大小相应结构体信息,
	T_U8 *DriverCnt: 此移动储存设备上总共有的分区数目
返回值说明	操作成功返回系统为移动储存设备分配的第一个分区的 ID
注意事项	目前内部输入(MEDIUM_SD == pDevInfo->nMainType || MEDIUM_USBHOST == pDevInfo->nMainType)是设计把一个 page 模拟成 32 个 sector
调用示例	StartDriver = FS_MountMemDev(pDevInfo, &DriverCnt);

3.3.4 FS_UnMountMemDev

原　　型	T_BOOL FS_UnMountMemDev(T_U8 DriverID);
功能概述	将卸载相应移动储存设备在 MOUNT 时所申请的内存,之后将不能使用这个此移动储存设备所对应的分区的文件系统与 U 盘功能.
参数说明	T_U8 DriverID:盘符 ID,当此移动储存设备加载的时候,系统分配给移动储存设备的分区的 ID.
返回值说明	操作成功返回 1,否则返回 0.
注意事项	只有在不使用此移动储存设备功能后才可调用

原　　型	T_BOOL FS_UnMountMemDev(T_U8 DriverID);
调用示例	FS_UnMountMemDev(DriverID);

3.3.5 FS_Destroy

原　　型	T_BOOL FS_Destroy();
功能概述	当系统关机时,不再使用文件系统功能时,调用此函数,之后将不能再使用文件系统功能,它将释放 NANDFLASH,SD 卡,全部卸载,此功能只在关机最后使用.
参数说明	无
返回值说明	操作成功返回 1,否则返回 0.
注意事项	此操作将会关闭所有打开的文件,并回写 NANDFLASH,并释放所有模块申请过的内存,将不能再使用此模块提供的任何功能.
调用示例	FS_Destroy();

3.3.6 FS_UnInstallDriver

原　　型	T_U8 FS_UnInstallDriver(T_U8 DriverID, T_U8 DriverCnt);
功能概述	卸载所有对应的分区中的文件系统层信息,但是此函数被调用时,如果分区所在的 MTD 没有被加载的话,此时将会加载 MTD,再加载分区
参数说明	T_U8 DriverID: 相应分区的 ID
	T_U8 DriverCnt:总共要卸载分区的数目
返回值说明	返回已经卸载的分区数目
注意事项	此操作如果进入 U 盘的时候调用,保证加载所对应的分区,此次加载,如果系统之前没有被使用的话,有可能有一定的延时
调用示例	FS_UnInstallDriver(DriverID, DriverCnt);

3.3.7 FS_InstallDriver

原　　型	T_U8 FS_InstallDriver(T_U8 DriverID, DriverCnt);
功能概述	加载逻辑分区，也就是初始化相应盘,它将会加载相应分区信息,这个为了某些应用,可以延时,又不想对后续的某个操作加载分区时使用.

原　　型	T_U8 FS_InstallDriver(T_U8 DriverID, DriverCnt);
参数说明	T_U8 DriverID: 相应分区的 ID
	T_U8 DriverCnt:总共要加载分区的数目
返回值说明	返回成功加载的分区数目
注意事项	可能会有一定的延时,因为此时将加载分区信息,如果所对应的 MTD 没有被加载的话,将会先加载 MTD,再加载分区. 多线程：fs install driver, notice: only insert driver ID to install queue, need wait for thead auto install，而单线程的话就直接进行加载
调用示例	FS_InstallDriver(DriverID, DriverCnt);

3.3.8 FS_InstallDriver_CallBack

原　　型	T_U8 FS_InstallDriver(T_U8 DriverID, T_U8 DriverCnt, F_GetDriverCallback pGet_Griver)
功能概述	此接口和上面接口相同，主要是增加一个参数，此参数是作为外面一个回调函数传进来，功能是在执行此函数时，外面也可以同时执行传进来的回调的函数 pGet_Griver。
参数说明	T_U8 DriverID 分区的 ID
	T_U8 DriverCnt:总共要加载分区的数目
	F_GetDriverCallback pGet_Griver 回调函数
返回值说明	返回成功加载的分区数目
注意事项	可能会有一定的延时,因为此时将加载分区信息,如果所对应的 MTD 没有被加载的话,将会先加载 MTD,再加载分区. 多线程：fs install driver, notice: only insert driver ID to install queue, need wait for thead auto install，而单线程的话就直接进行加载
调用示例	typedef T_BOOL (*F_GetDriverCallback)(T_VOID); FS_InstallDriver(T_U8 DriverID, T_U8 DriverCnt, F_GetDriverCallback pGet_Griver)

3.3.9 FS_GetDriver

原　　型	T_BOOL FS_GetDriver(PDRIVER_INFO pDriverInfo, T_U8 DriverID);
功能概述	得到所对应分区的一些信息,以便在 USB 加载盘的时候使用。
参数说明	PDRIVER_ INFO pDriverInfo:输出的对应分区的信息
	T_U32 DriverID: 要得到所对应分区的 ID
返回值说明	操作成功返回 1,否则返回 0.
注意事项	无
调用示例	FS_GetDriver(pDriverInfo, DriverID);

3.3.10 FS_GetFirstDriver

原　　型	T_BOOL FS_GetFirstDriver(PDRIVER_INFO pDriverInfo);
功能概述	得到第一个所对应分区的一些信息,以便在 USB 加载盘的时候使用。
参数说明	PDRIVER_INFO pDriverInfo:输出的对应分区的信息
返回值说明	操作成功返回 1,否则返回 0.
注意事项	无
调用示例	FS_GetFirstDriver(pDriverInfo);

3.3.11 FS_GetNextDriver

原　　型	T_BOOL FS_GetNextDriver(PDRIVER_INFO pDriverInfo);
功能概述	得到下一个所对应分区的一些信息,以便在 USB 加载盘的时候使用。
参数说明	PDRIVER_INFO pDriverInfo:输出的对应分区的信息
返回值说明	操作成功返回 1,否则返回 0.如果失败证明已经没有可加载的分区了,
注意事项	与 FS_GetFirstDriver 一起使用

原　　型	T_BOOL FS_GetNextDriver(PDRIVER_INFO pDriverInfo);

调用示例	DRIVER_INFO DriverInfo;
	T_BOOL Ret;
	Ret = FS_GetFirstDriver(&DriverInfo);
	While(AK_TRUE == Ret)
	{
	If(MEDIUM_NANDFLASH == DriverInfo.MainType)
	{
	//it is the partition of the nandflash
	If(SYSTEM_PARTITION == DriverInfo.SubType)
	{
	// it is the system partition
	}
	Else If(USER_PARTITION == DriverInfo.SubType)
	{
	// it is the user partition
	}
	}
	Ret = FS_GeNextDriver(&DriverInfo);
	}

3.3.12 FS_ChkDsk

原型	T_BOOL FS_ChkDsk(T_U8 DriverID, F_ChkDskCallback pCallBack, T_VOID *CallbackData)
功能概述	当系统发现掉电后,将调用此功能进行检查对对应分区的完整性,并修复错误
参数说明	T_U8 DriverID :对应分区 ID,
	F_ChkDskCallback pCallBack: 系统的回调函数,此函数将会在完成的百分比变化时调用,精度为 1% Typedef void F_ChkDskCallback(T_VOID *pData);
	T_VOID *CallbackData: 回调函数的参数

原型	T_BOOL FS_ChkDsk(T_U8 DriverID, F_ChkDskCallback pCallBack, T_VOID *CallbackData)
返回	如果有对应的 DriverID,将检查并返回 TRUE,否则返回 FALSE
注意事项	这个类似 CHKDSK 命令,将会有一定的耗时,只在系统认定是异常掉电才使用.
调用示例	FS_ChkDsk(0, pCallBackFunc, pData);

3.3.13 FS_FlushDriver(T_U8 DriverID)

原型	T_BOOL FS_ FlushDriver(T_U8 DriverID)
功能概述	将分区的缓冲数据全部写入到 NANDFLASH 上.此操作在 U 盘方式如果没有数据写入时调用
参数说明	T_U8 DriverID :对应分区 ID,
返回	
注意事项	
调用示例	FS_FlushDriver (0);

3.3.14 FS_GetAsynBufInfo

原　　型	T_BOOL FS_GetAsynBufInfo(T_U32 *UseSize, T_U32 * BufSize)
功能概述	获取异步写 buffer 的内存应用情况, 可以用来计算 buffer 目前的状态
参数说明	UseSize 输出 buffer 已占用的字节数
	BufSize 输出 buffer 总的可用字节数
返回值说明	操作成功返回 AK_TRUE,否则返回 AK_FLASE.
注意事项	无
调用示例	

3.3.15 FS_GetFakeMedium

原　　型	T_PMEDIUM FS_GetFakeMedium(T_U8 DriverID);
功能概述	通过 driverID 获取分区相应的介质 Medium
参数说明	T_U8 DriverID 分区的 ID

原　　型	T_PMEDIUM FS_GetFakeMedium(T_U8 DriverID);
返回值说明	操作成功返回 T_PMEDIUM 的结构体
注意事项	无
调用示例	Medium = FS_GetFakeMedium(0);

3.3.16 FS_CheckInstallDriver

原　　型	T_BOOL FS_CheckInstallDriver(T_U8 DriverName);
功能概述	检测分区是否加载成功
参数说明	T_U8 DriverName 分区的盘符名
返回值说明	操作成功返回成功与否
注意事项	无
调用示例	FS_CheckInstallDriver("A");

3.3.17 FS_SetAsynWriteBufSize

原　　型	T_BOOL FS_SetAsynWriteBufSize(T_U32 BufSize, T_U8 DriverID);
功能概述	设置异步写的 BUF 大小，如果 BufSize 传是 0，那么就是关闭异步写，否则就是打开。
参数说明	T_U32 BufSize　异步写的 BUF 大小
	T_U8 DriverID　分区的 ID
返回值说明	成功与否
注意事项	一般以 64K 的数据的部数增加。单位以字节

调用示例	FS_SetAsynWriteBufSize(64*1024, 0); //打开异步写
	FS_SetAsynWriteBufSize(0, 0); //关闭异步写

3.3.18 FS_QuickFormatDriver

原　　型	T_BOOL FS_QuickFormatDriver(T_U8 DriverID);
功能概述	通过分区的 ID 进行快速格式化分区，此接口主是对原有的分区进行格式化，保留原有的文件系统。
参数说明	T_U8 DriverID 分区的 ID
返回值说明	成功与否
注意事项	此接口在这提供，主要是文件系系统的 driver 层接口不向外提代，所以从这里提供出去。
调用示例	FS_QuickFormatDriver(0);

3.3.19 FS_FormatDriver

原　　型	T_BOOL FS_FormatDriver(T_U8 DriverID, E_FATFS FsType);
功能概述	通过分区的 ID 进行快速格式化分区,同时可以格式成指定的文件系统类型
参数说明	T_U8 DriverID 分区的 ID
	E_FATFS FsType 文件系统的类型 typedef enum { FAT_FS_ERROR = 0, FAT_FS_16 = 1, FAT_FS_32 = 2, FAT_FS_EX = 3 }E_FATFS;
返回值说明	成功与否
注意事项	此接口在这提供，主要是文件系系统的 driver 层接口不向外提代，所以从这里提供出去。
调用示例	FS_FormatDriver(0, 2);

3.3.20 FS_GetDriverCapacity

原　　型	T_U32 FS_GetDriverCapacity(T_U8 DriverID, T_U32 *high);
功能概述	获取分区的总容量大小
参数说明	T_U8 DriverID 分区的 ID
	T_U32 *high 高位，如：大小==4G 的，就返回 1。
返回值说明	分区的总容量大小
注意事项	此接口在这提供，主要是文件系系统的 driver 层接口不向外提代，所以从这里提供出去。
调用示例	FS_GetDriverCapacity(0, &high);

3.3.21 FS_GetDriverUsedSize

原　　型	T_U32 FS_GetDriverUsedSize(T_U8 DriverID, T_U32 *high);
功能概述	获取分区的已用空间的大小
参数说明	T_U8 DriverID 分区的 ID
	T_U32 *high 高位，如：大小==4G 的，就返回 1。
返回值说明	已用空间的大小
注意事项	此接口在这提供，主要是文件系系统的 driver 层接口不向外提代，所以从这里提供出去。
调用示例	FS_GetDriverUsedSize(0, &high);

3.3.22 FS_GetDriverFreeSize

原　　型	T_U32 FS_GetDriverFreeSize(T_U8 DriverID, T_U32 *high);
功能概述	获取分区的可用空间大小
参数说明	T_U8 DriverID 分区的 ID
	T_U32 *high 高位，如：大小==4G 的，就返回 1。
返回值说明	可用空间大小
注意事项	此接口在这提供，主要是文件系系统的 driver 层接口不向外提代，所以从这里提供出去。

原　　型	T_U32 FS_GetDriverFreeSize(T_U8 DriverID, T_U32 *high);
调用示例	FS_GetDriverFreeSize(0, &high);

3.3.23 FS_GetVersion

原　　型	T_U8 *FS_GetVersion(T_VOID);
功能概述	获取 mount 库的版本号
参数说明	无
返回值说明	mount 库的版本号
注意事项	
调用示例	

3.3.24 FS_LowFormat

原　　型	T_BOOL FS_LowFormat(T_FS_PARTITION_INFO PartitionInfo[], T_U32 nNumPartion, T_U32 resvSize, T_U32 StartBlock, PFORMAT_INFO pFormatInfo);
功能概述	Mount 盘时，并进行格式化磁盘，此接口一般都是烧录时使用到
参数说明	T_FS_PARTITION_INFO PartitionInfo[] 分区信息
	T_U32 nNumPartion 分区的个数
	T_U32 resvSize　　保留大小
	T_U32 StartBlock　分区的开始位置
	PFORMAT_INFO pFormatInfo 介质信息
返回值说明	mount 库的版本号
注意事项	
调用示例	

3.3.25 FS_CreateCache

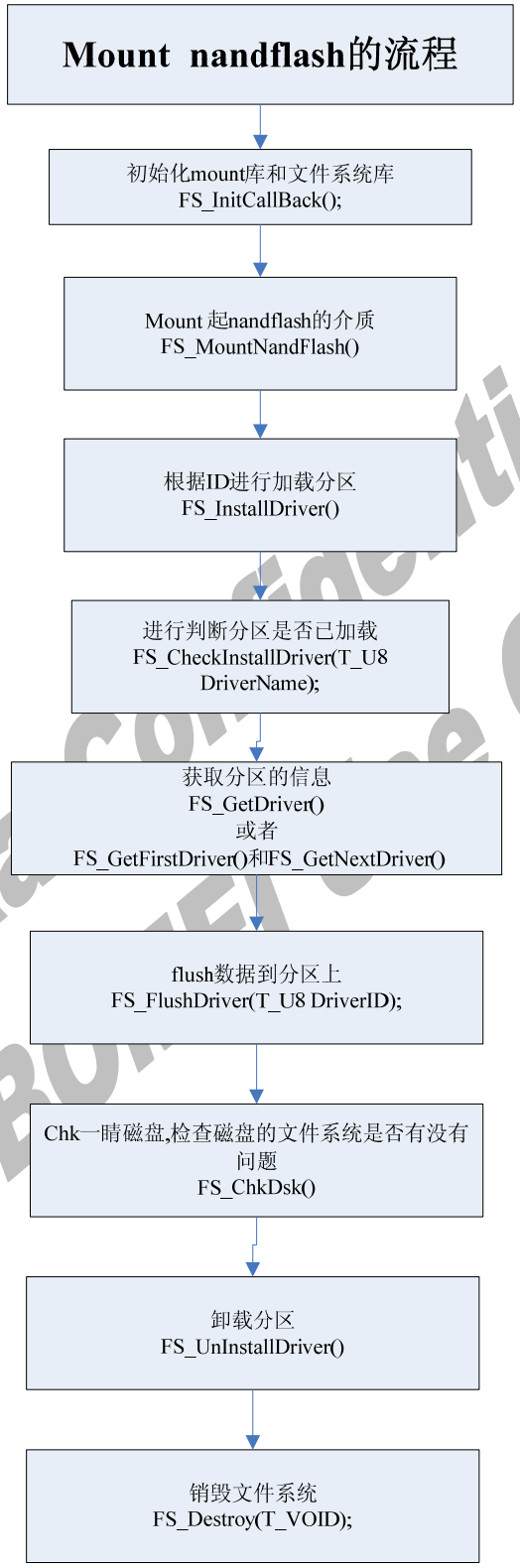
原　　型	T_PMEDIUM FS_CreateCache(T_PMEDIUM medium, T_U32 CacheSize)
功能概述	创建 Cache
参数说明	T_PMEDIUM medium 分区的介质
	T_U32 CacheSize　cache 的大小，以字节为单位
返回值说明	新创建出来的 medium,或失败为空
注意事项	与 FS_DestroyCache()配对使用，否则出现内存泄漏。
调用示例	Pmedium = FS_CreateCache(T_PMEDIUM medium, T_U32 CacheSize) FS_DestroyCache(Pmedium);

3.3.26 FS_DestroyCache

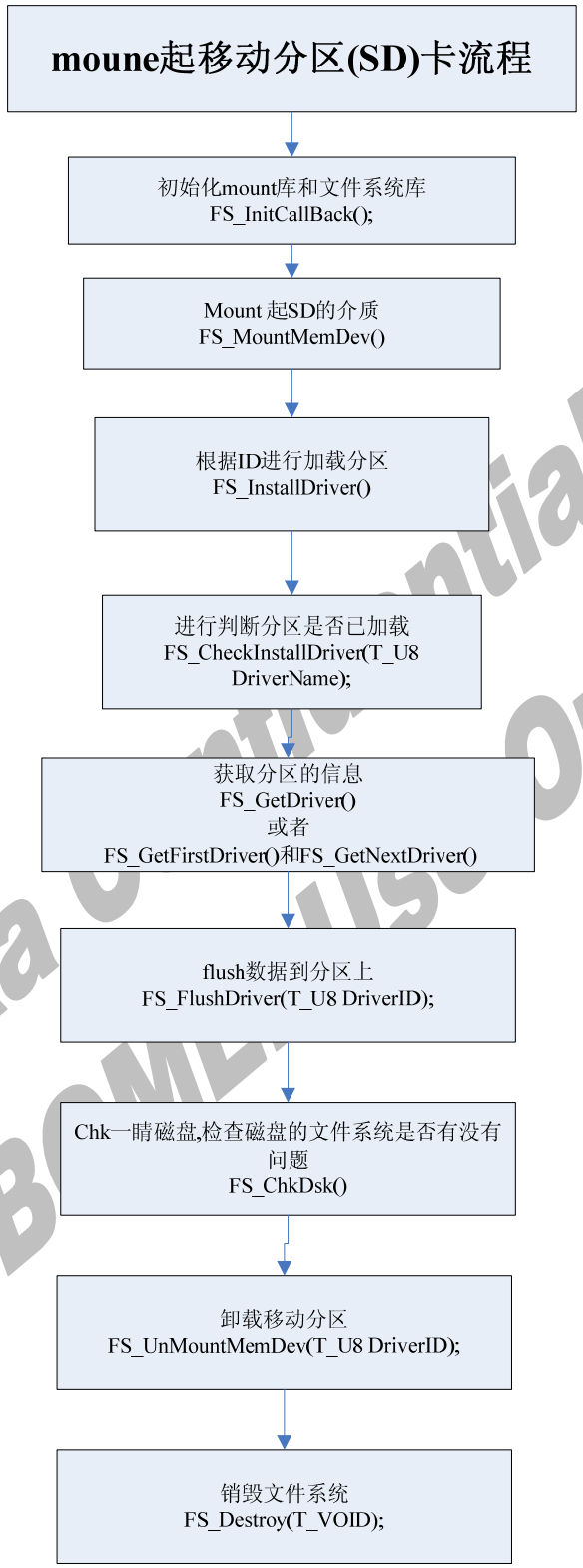
原　　型	T_VOID FS_DestroyCache(T_PMEDIUM CacheMedium);
功能概述	销毁 cache
参数说明	T_PMEDIUM CacheMedium 创建出来的新 medium
返回值说明	无
注意事项	
调用示例	Pmedium = FS_CreateCache(T_PMEDIUM medium, T_U32 CacheSize) FS_DestroyCache(Pmedium);

3.4 全局函数的调用流程

3.4.1 Mount nand 的流程



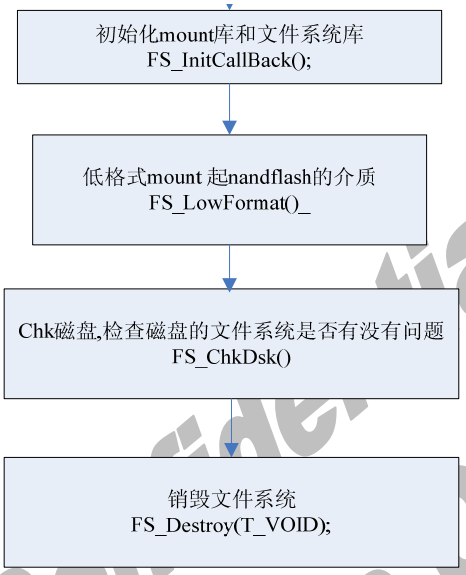
3.4.2 Mount SD 的流程



3.4.3 低格式进行mount起nand或sd卡的流程

此流程同样是 mount 起盘，不过这个流程可以在 mount 盘的过程中进行格式化，此接口一般是烧录操作才使用。

低格式进行 mount 起 nand 的流程：



低格式进行 mount 起 sd 卡的流程：