

目录

系统函数说明 V1.57.....	6
一、类型定义	6
二、常量定义	6
键盘键值定义.....	7
文件系统.....	7
数据库.....	7
串口设备.....	8
输入法.....	8
键盘.....	9
图标.....	9
三、结构体类型定义	12
sFILE	12
sDBF.....	12
sRTC.....	13
sUART.....	13
sMENU.....	13
四、函数定义	14
数据库操作函数	14
DbfRecordRead.....	14
DbfRecordWrite	14
DbfOpen	14
DbfClose.....	15
DbfGotoRecord.....	15
DbfGetCurrentRecord.....	15
DbfRecordSize	16
DbfRecordAppend.....	16
DbfRecordCount.....	16
DbfRecordIsDeleted	17
DbfRecordLocate	17
DbfGetMatchCount	17
DbfFieldSize	18
DbfFieldGet.....	18
DbfFieldSet.....	19
DbfFieldCount.....	19
DbfFieldInfo	19
DbfRecordDelete	20
DbfRecordRestore.....	20
DbfCreate	20
DbfCopy.....	21
DbfRecordErase	21
DbfFieldIndex.....	22
DbfRecordPack.....	22
DbfDelete.....	22
DbfRecordClear.....	22
文件操作函数	23
FileDelete	23
FileChoiceRecycle.....	23
FileClose.....	23
FileRead	24
FileWrite.....	24
FileSeek	24
FileLength.....	25
FileTell.....	25
FileOpen.....	25

FileOpenExpand.....	26
FileGetCreateTime	26
FileGetType	26
输入法函数	27
InputSetParam.....	27
Input.....	27
InputBh	28
InputMix	28
InputYw.....	28
InputPy	29
InputSz.....	29
InputPassword.....	29
InputSzr.....	30
键盘函数	30
KeyValue.....	30
KeyWait.....	30
KeySleepWait	31
KeyTimeWait	31
KeyScan	31
KeySendValue	31
SetPowerOffFun	32
SetPowerOnFun	32
SetKeyFuntion	32
显示函数	33
LcdClear.....	33
LcdSetDot	33
LcdSetArea.....	33
LcdSetColor.....	33
LcdSaveArea.....	34
LcdRestoreArea	34
LcdDrawLine.....	34
LcdDrawRect.....	35
LcdDisplayIcon.....	35
LcdPutChar	35
LcdPutNChar.....	36
LcdPutString.....	36
LcdPrintf.....	36
LcdMoveto	37
LcdSetCursor.....	37
LcdDrawArcAngle	38
LcdDrawArcCenter.....	38
LcdGetIconAddr	38
LcdGetX.....	39
LcdGetY.....	39
LcdSetBgColor	39
RGB888ToRGB565.....	39
LcdGetColor	39
菜单函数	40
MenuDisplay.....	40
MenuTextMain	40
MenuDisplayCenter.....	41
MenuDisplayIcon.....	41
MenuGraphMain	42
延时函数	42
DelayMicrosecond.....	42
DelayMillisecond.....	43
串口函数	43
UartDeviceInit.....	43
UartOpen.....	43
UartClose	43
UartRead	44
UartWrite.....	44
UartRxByte	44

UartTxByte	45
UartRxString.....	45
UartTxString.....	45
UartPrintf.....	46
变量函数	46
ParamRead.....	46
ParamDelete.....	46
ParamWrite.....	47
ParamEarse.....	47
ParamSetWord.....	47
ParamGetWord	48
时间函数	48
RtcSetTime	48
RtcSetAlarm.....	48
RtcSetDate	49
RtcGetDate	49
RtcGetWeek.....	49
RtcGetTime	49
RtcGetDateString.....	50
RtcSetDateString.....	50
RtcGetTimeString.....	50
RtcSetTimeString.....	50
RtcDisplayOn	50
RtcDisplayOff.....	51
C 库函数.....	51
Lib_puts	51
Lib_printf.....	51
Lib_sprintf.....	52
Lib_atof.....	52
Lib_atoi.....	52
Lib_atol.....	53
Lib_srand.....	53
Lib_rand	53
Lib_calloc.....	53
Lib_free.....	53
Lib_malloc.....	53
Lib_realloc.....	54
调试函数	54
Bell.....	54
LedLeft.....	54
LedMiddle.....	54
BackLightLcd.....	55
BackLightLed.....	55
BackLightKey	55
DetectVoltage.....	55
DetectTemperature.....	55
BatBarDisplayON.....	55
BatBarDisplayOff.....	56
PowerManage.....	56
ProgramLock.....	56
UsbSendStringToPC	57
USB 通信函数.....	57
UsbSendStringToPC	57
UsbSendKeyToPC.....	57
UsbSendAsciiToPC.....	57
UsbConnectToPC	58
UsbDisconnectToPC.....	58
UsbMain	58
VirtualComOpen.....	58
VirtualComClose	58
VirtualComWrite.....	59
VirtualComRead	59
VirtualComRxString.....	59

VirtualComTxString	59
SetUsbFuntion	59
字符编码转换	61
EncGetUtf8Size	61
EncUtf82UnicodeOne	61
EncUtf82Unicode	61
EncUtf82UnicodeStr	62
EncUnicode2Utf8One	62
EncUnicode2Utf8	63
EncUnicode2Utf8Str	63
EncUtf82GbkStr	64
EncUnicode2GbkStr	64
EncUnicode2GbkOne	65
EncGbk2UnicodeOne	65
Ascii2Hex	65
Hex2Ascii	65
LowerCase	66
UpperCase	66
TrimString	66
缓冲区 SQL 函数	66
SqlOpen	66
SqlGotoRecord	67
SqlRecordRead	67
SqlRecordCount	67
SqlFieldCount	67
SqlFieldInfo	67
SqlFieldGet	67
TF/SD 卡读写程序接口	68
FatMount	69
FatOpen	70
FatClose	73
FatRead	74
FatWrite	75
FatLseek	76
FatTruncate	78
FatSync	11
FatOpenDir	12
FatReadDir	13
FatGetfree	15
FatStat	16
FatMkdir	17
FatUnlink	18
FatChmod	19
FatUtime	20
FatRename	21
FatChdir	22
FatChdrive	23
FatGetcwd	24
FatForward	25
FatMkfs	27
FatFdisk	28
FatGets	30
FatPutc	31
FatPuts	31
FatPrintf	32
FatTell	33
FatEof	34
FatSize	35
FatError	36
五、外设功能	37
1、Barcode 条码(一维、二维)	37
2、Bluetooth 蓝牙	37

3、Camera 摄像.....	38
4、GPRS 通信功能.....	38
5、GPS 全球定位系统.....	40
6、IRDA_H 高速红外.....	40
7、IRDA_L 载波红外 38.4K.....	40
8、RFID 低频电子标签.....	41
9、RFID 高频频电子标签.....	41
ISO14443A 数据读写函数.....	43
1、 初始化端口.....	46
2、 关闭端口.....	47
3、 指定设备标识.....	47
4、 获取设备标识.....	47
5、 读取硬件版本号.....	48
6、 读写器工作模式.....	48
7、 设置天线状态.....	48
8、 寻 Mifare 卡.....	49
9、 Mifare 卡防冲突.....	49
10、 Mifare 选卡.....	50
11、 休眠 Mifare 卡.....	50
12、 Mifare 卡认证.....	50
13、 Mifare 卡读卡.....	51
14、 Mifare 写卡.....	51
15、 Mifare 钱包初始化(暂无).....	52
16、 读取 Mifare 钱包值(暂无).....	52
17、 Mifare 钱包扣款(暂无).....	52
18、 Mifare 钱包增值(暂无).....	53
19、 Mifare 数据回传(暂无).....	53
20、 Mifare 数据传送(暂无).....	54
21、 Utri Light 选卡.....	54
22、 Utri Light 写卡.....	54
23、 ISO15693_Inventory.....	55
24、 ISO15693_Stay_Quiet.....	55
25、 选择 15693.....	55
26、 ISO15693_ResetToReady.....	56
27、 读取 15693 中的数据.....	56
28、 写入数据到 15693 卡中.....	57
29、 锁定 15693 卡某块.....	57
30、 写入 15693 的 AFI.....	58
31、 锁定 15693 的 AFI.....	58
32、 写入 15693 的 DSFID.....	58
33、 写入 15693 的 DSFID.....	59
34、 读取 15693 的卡片信息.....	59
35、 读取 15693 的某块数据信息.....	60
1 API 返回值.....	60
10、RFID 超高频频电子标签.....	61
11、RS232 串行数据接口.....	61
12、RS485 串行数据接口.....	61
13、Voice 中文语音播读.....	62
14、WIFI(Wireless Fidelity).....	62
六、硬件函数	62
1、中断服务.....	62
BSP_IntVectSet.....	62
2、秒定时调用.....	63
SetSecondIRQ.....	63
3、UART.....	64

4、SPI..... 64

5、IIC..... 64

6、ADC..... 64

7、DAC..... 64

8、TIM..... 64

9、SDIO..... 64

10、JTAG..... 64

11、IO..... 64

七、OS 多任务系统..... 65

 1、参考OS 中文说明书.pdf..... 65

八、GUI 图形界面..... 65

 1、参考GUI 中文说明书.pdf..... 65

九、附录..... 65

 GB2312 符号表..... 65

 KEYBOARD/KEYPAD 键值表..... 70

系统函数说明 V1.57

一、类型定义

为了方便使用C或者C++的数据类型，将常用的数据类型定义如下：

```
typedef unsigned int      U32;
typedef unsigned short    U16;
typedef unsigned char     U8;
typedef int               S32;
typedef short int         S16;
typedef char              S8;
```

二、常量定义

已经定义的常量为系统内部已经使用，用户不要修改

键盘键值定义

调用键盘函数，如KeyWait，返回值为以下值，其中KEY_SCAN为保留值，并无按键与之对应

```
#define KEY_NUM0          0x30          //对应键盘“0”按键
#define KEY_NUM1          0x31          //对应键盘“1”按键
#define KEY_NUM2          0x32          //对应键盘“2”按键
#define KEY_NUM3          0x33          //对应键盘“3”按键
#define KEY_NUM4          0x34          //对应键盘“4”按键
#define KEY_NUM5          0x35          //对应键盘“5”按键
#define KEY_NUM6          0x36          //对应键盘“6”按键
#define KEY_NUM7          0x37          //对应键盘“7”按键
#define KEY_NUM8          0x38          //对应键盘“8”按键
#define KEY_NUM9          0x39          //对应键盘“9”按键
#define KEY_DOT            0x2E          //对应键盘“*”按键
#define KEY_SWITCH        0x2D          //对应键盘“#”按键
#define KEY_LAMP           0x15          //对应键盘“背光”按键或者Tab键
#define KEY_SCAN           0x16          //无对应键盘按键
#define KEY_ENTER          0x0D          //对应键盘“确定”按键或者Ent键
#define KEY_ESC            0x1B          //对应键盘“退出”按键或者Esc键
#define KEY_UP             0x11          //对应键盘“↑”按键
#define KEY_DOWN           0x12          //对应键盘“↓”按键
#define KEY_LEFT           0x13          //对应键盘“←”按键
#define KEY_RIGHT          0x14          //对应键盘“→”按键
#define KEY_DELETE         0x08          //对应键盘“删除”按键或者Del键
#define KEY_FUN            0x0A          //对应键盘“功能”按键或者Fn键
#define KEY_SIDE_R         0x1c          //对应键盘右侧按键
#define KEY_SIDE_L         0x1d          //对应键盘左侧按键
#define KEY_FUN1           0x1E          //对应键盘的F1功能
#define KEY_FUN2           0x1F          //对应键盘的F2功能
```

文件系统

当前文件系统文件名长度不能超过31个ASCII字符，超过部分将被截断。

注意：一个中文占两个ASCII字符位置

```
#define FILE_NAME_LENGTH  32
#define SEEK_SET           0           /*文件定位标志:从文件头*/
#define SEEK_CUR           1           /*文件定位标志:从当前位置*/
#define SEEK_END           2           /*文件定位标志:从文件尾*/
```

数据库

数据库目前规定单个记录最大字段数目为100个，如有单个记录超过100个字段的需要，请提出要求。

```
#define MAX_FIELD_NUM      100
#define DBF_OPER_OK        0
#define DBF_INVALID_FORMAT -1
#define DBF_FILE_FAILED    -2
#define DBF_NO_RECORD      -3
#define DBF_NO_FIELD       -4
```

串口设备

系统可以和多种串口设备通信，串口支持7，8，9位数据位通信。

```
#define DEVICE_UART_TTL          1          //普通串口
#define DEVICE_LOW_IRDA          2          //低速红外
#define DEVICE_HIGHT_IRDA        3          //高速红外
#define DEVICE_SCANNER            4          //条码扫描
#define DEVICE_GPRS_MODEM         5          //GPRS通讯
#define DEVICE_CDMA_MODEM         6          //CDMA通讯
#define DEVICE_RS485              7          //RS485通讯
#define DEVICE_RS232              8          //RS232通讯
#define DEVICE_WIRELESS           9          //无线模块
#define DEVICE_BLUETOOTH          10         //蓝牙
#define DEVICE_WIFI               11         //无线局域网
#define DEVICE_RFID_LOW           12         //低频射频标签
#define DEVICE_RFID_HIGH          13         //高频射频标签
#define DEVICE_RFID_UHIGH         14         //超高频射频标签
#define DEVICE_VOICE              15         //语音
#define DEVICE_VEDIO              16         //摄像
#define DEVICE_WIRELESS433        17         //433功能
#define DEVICE_RFID_24G           18         //2.4GRFID
#define DEVICE_ZIGBEE             19         //Zigbee通信
#define DEVICE_GPS                20         //GPS功能
#define DEVICE_MBUS               21         //MBUS功能
#define DEVICE_VIRTUAL_COM        22         //虚拟串口

#define DEVICE_COMMAND_MODE       0          //工作在命令模式
#define DEVICE_DATA_MODE          1          //工作在数据模式
#define DEVICE_STATE_MODE         2          //读取状态模式
#define DEVICE_STATE_MODE1        3          //读取状态模式1

#define UART_MODE_7B_ODD_1S        1          //7位数据位，奇校验，一位停止位
#define UART_MODE_7B_EVEN_1S      2          //7位数据位，偶校验，一位停止位
#define UART_MODE_8B_NONE_1S      3          //8位数据位，无校验，一位停止位
#define UART_MODE_8B_ODD_1S       4          //8位数据位，奇校验，一位停止位
#define UART_MODE_8B_EVEN_1S      5          //8位数据位，偶校验，一位停止位
#define UART_MODE_9B_NONE_1S      6          //9位数据位，无校验，一位停止位
```

输入法

系统具有强大的输入法，能够支持数字，英文，中文输入，中文输入法支持拼音输入和笔画输入，汉字库标配GB2312字库，可以选用GBK字库

```
#define INPUT_UNLOCKED           0          //解除锁定
#define INPUT_LOCKED             1          //锁定输入法

#define INPUT_TYPE_NUMBER        0          //数字输入
#define INPUT_TYPE_CHINESEPY     1          //中文拼音输入
```



```
#define INPUT_TYPE_CHINESEBH 2 //中文笔画输入
#define INPUT_TYPE_MIX 3 //数字字母混合输入
#define INPUT_TYPE_CAPTITAL 4 //大小字母输入
#define INPUT_TYPE_LOWERCASE 5 //小写字母输入
```

键盘

```
#define Keyboard_LeftControl 0x01
#define Keyboard_LeftShift 0x02
#define Keyboard_LeftAlt 0x04
#define Keyboard_Left_GUI10 0x08
#define Keyboard_RightControl 0x10
#define Keyboard_RightShift 0x20
#define Keyboard_RightAlt 0x40
#define Keyboard_Right_GUI10 0x80

#define Keyboard_Num_Lock 0x01
#define Keyboard_Caps_Lock 0x02
#define Keyboard_Scroll_Lock 0x04
```

图标

系统内置了多种图标以便实现图形化，并提高了图标显示函数，如用户需要特殊的图标，可以提出，我们将帮助用户完成设计

坐标系以左上角为零点，横向为X轴，轴向为Y轴，只有一个象限

```
#define LCD_X_PELS 灰度屏160 彩屏240 //屏幕X轴方向像素数目
#define LCD_Y_PELS 灰度屏160 彩屏320 //屏幕Y轴方向像素数目
#define LCD_LINE_MIN 灰度屏0 彩屏0 //最小行号
#define LCD_LINE_MAX 灰度屏159 彩屏239 //最大行号
#define LCD_ROW_MIN 灰度屏0 彩屏0 //最小列号
#define LCD_ROW_MAX 灰度屏159 彩屏320 //最大列号
```



```
#define ICON_RUN_SPECIFY 0
```



```
#define ICON_RUN_SELETE 1
```



```
#define ICON_SHOW_INFO 2
```







































```
#define ICON_COM_MANAGE 3
```



```
#define ICON_FILE_MANAGE 4
```

		#define ICON_SYS_SETTING	5
		#define ICON_CALENDAR	6
		#define ICON_CALCULATE	7
		#define ICON_HELP	8
		#define ICON_USB	9
		#define ICON_USART	10
		#define ICON_BULETOOTH	11
		#define ICON_HIGHT_IRDA	12
		#define ICON_LOW_IRDA	13
		#define ICON_WIRELESS	14
		#define ICON_PASSWORD	15
		#define ICON_ALARM	16
		#define ICON_CONTRAST	17
		#define ICON_TIME	18
		#define ICON_USER_INFO	19
		#define ICON_DISPLAY_MODE	20
		#define ICON_SHUTDOWN	21
		#define ICON_EXE_FILE	22

		#define ICON_DBF_FILE	23
		#define ICON_TXT_FILE	24
		#define ICON_WAV_FILE	25
		#define ICON_OTHER_FILE	26
		#define ICON_ALL_FILE	27
		#define ICON_CHECK_FILE	28
		#define ICON_RECIRLCE_FILE	29
		#define ICON_ENTERPRISE	30
		#define ICON_USER_NAME	31
		#define ICON_USER_INDEX	32
		#define ICON_MACH_INDEX	33
		#define ICON_REMARKS	34
		#define ICON_LAMP	35
		#define ICON_ASSISTANT	36
		#define ICON_READER	37
		#define ICON_SIGH	38
		#define ICON_RIGHT	39
		#define ICON_ERROR	40

		#define ICON_FIND	41
		#define ICON_BELL	42
		#define ICON_BARCODE	43

三、结构体类型定义

结构体类型大多数情况下只用来定义数据类型，而结构体内部成员所表示的意义一般不需要理会，菜单类型例外。

sFILE

```
typedef struct {
    S8 FileName[FILE_NAME_LENGTH];
    U8 FileFlagRw;
    U16 FirstBlockNo;
    U16 BlockNo;
    U32 FileSize;
    U8 *pData;
}sFILE;
```

sDBF

sDbfHead

```
typedef struct {
    U8 cDbfVer;
    U8 cYear;
    U8 cMonth;
    U8 cDay;
    U32 iRecordCount;
    U16 iFirstRecordOffset;
    U16 iRecordLength;
    U8 szReserved1[16];
    U8 cDbfFlag;
    U8 cCodePage;
    U8 szReserved2[2];
}sDbfHead;
```

sField

```
typedef struct {
```

```

    S8  szFieldName[11];
    U8  cFieldType;
    U32 iFieldOffset;
    U8  cFieldLength;
    U8  cDecimalNum;
    U8  szReserved[14];
} sField;

typedef struct {
    sFILE stDbfFile;
    sDbfHead stDbfHead;
    U32 iCurrentRecord;
    U16 iFieldNum;
    sField stDbfField[MAX_FIELD_NUM];
    U8 cSum;
} sDBF;

```

sRTC

```

typedef struct {
    U8 year;
    U8 month;
    U8 day;
    U8 hour;
    U8 min;
    U8 sec;
} sRTC;

```

sUART

```

typedef struct {
    U8 device;
    U8 mode;
    U32 baud;
    U8 state;
    U8 port;
    void *pUsartx;
} sUART;

```

sMENU

```

typedef struct {
    U8 startx;           //mode=0, 2有效
    U8 starty;           //mode=0, 2有效
    U8 endx;             //mode=0时有效
    U8 endy;             //mode=0时有效
    S8 *hotkey;          //热键指针，例如 "12345"，不需要时要赋空串
    U16 amount;          //总的要显示的菜单数目
    U8 count;            //单页可以显示的菜单数目，mode=1或2时有效

```

```

U8 border;           //菜单边框 当boder=1时，边界向外扩8个像素
U16 current;         //当前选定的菜单
U8 mode;             //0：固定区域显示 1：中央区域显示 2：定位方式
                     //4：主菜单模式 5：从菜单模式
U8 character;        //一行显示字符的个数 偶数，当mode=0时有效
U8 style;            //使用的字体类型, 可选16、12、8
} sMENU;

```

四、函数定义

数据库操作函数

DbfRecordRead

S32 DbfRecordRead(U32 record, U8 *data, sDBF *dbf);

功能：读取指定记录的数据内容。

返回：返回DBF_OPER_OK表示设置成功；小于0表示错误。

参数：

- 1) record 要读取指定记录号。0 代表数据库的第 1 个记录。
- 2) data 存放记录内容的buffer。
- 3) dbf 已打开数据库的指针。

示例：

DbfRecordWrite

S32 DbfRecordWrite(U32 record, U8 *data, sDBF *dbf);

功能：写入指定记录的数据内容。

返回：返回DBF_OPER_OK表示设置成功；小于0表示错误。

参数：

- 1) record 要写入指定记录号。0代表数据库的第1个记录。
- 2) data 存放记录内容的buffer。
- 3) dbf 已打开数据库的指针。

示例：

DbfOpen

S32 DbfOpen(S8 *dbfname, sDBF *dbf);

功能： 打开一个数据库。当成功打开后，当前记录置于第1条记录（记录号为0）。

返回： 返回DBF_OPER_OK表示成功，其它值表示失败(小于0)。

参数：

1) **dbfname** 数据库名。

2) **dbf** 用于存放打开数据库的指针。用户需先声明一DBF的结构，并传递其地址进来。后续的数据库操作需要此指针。

示例：

```
sDBF dbf;
S32 iRet;
iRet = DbfOpen("test.dbf ", &dbf);
if(iRet < 0) {...failed...}
```

DbfClose

S32 DbfClose(sDBF *dbf);

功能： 关闭一个已打开的数据库。

返回： 返回DBF_OPER_OK表示成功，其它值表示失败(小于0)。

参数：

1) **dbf** 已打开数据库的指针。关闭后此指针不能再用于其它数据库操作。

示例：

```
S32 iRet;
DbfClose(&dbf);
if(iRet < 0) {...failed...}
```

DbfGotoRecord

S32 DbfGotoRecord(U32 record, sDBF *dbf);

功能： 定位到数据库的某一条记录。

返回： 返回DBF_OPER_OK表示成功，其它值表示失败(小于0)。

参数：

1) **record** 记录位置，0代表第一条数据库记录。

2) **dbf** 已打开数据库的指针。

示例：

```
S32 iRet;
iRet = DbfGotoRecord(10, &dbf); //定位到第11条记录。
if(iRet < 0) {...failed...}
```

DbfGetCurrentRecord

S32 DbfGetCurrentRecord(sDBF *dbf);

功能： 获取当前的记录位置。

返回： 返回值为记录位置。大于等于0表示正确返回记录位置，小于0表示失败。

参数:

- 1) dbf 已打开数据库的指针。

示例:

```
S32 iRet;  
    iRet = DbfGetCurrentRecord(&dbf);  
if(iRet < 0) {...failed...}
```

DbfRecordSize

`S32 DbfRecordSize(sDBF *dbf);`

功能: 获取数据库一条记录的长度。

返回: 返回值为记录长度。大于等于0表示正确返回记录长度，小于0表示失败。

参数:

- 1) dbf 已打开数据库的指针。

示例:

```
S32 iRet;  
    iRet = DbfRecordSize(&dbf);  
if(iRet < 0) {...failed...}
```

DbfRecordAppend

`S32 DbfRecordAppend(sDBF *dbf);`

功能: 在打开的数据库后面新增一条记录，并切换记录位置到新增的记录。

返回: 返回DBF_OPER_OK表示成功，其它值表示失败(小于0)。

参数:

- 1) dbf 已打开数据库的指针。

示例:

```
S32 iRet;  
    iRet = DbfRecordAppend(&dbf);  
if(iRet < 0) {...failed...}  
iRet = DbfFieldSet(0, "test", &dbf); //设置新添加记录的第一个字段值为test
```

DbfRecordCount

`S32 DbfRecordCount(sDBF *dbf);`

功能: 返回已打开的数据库的记录总数。

返回: 大于等于0表示成功，小于0表示失败。

参数:

- 1) dbf 已打开数据库的指针。

示例:

```
S32 iRet;  
    iRet = DbfRecordCount(&dbf);  
if(iRet < 0) {...failed...}
```


DbfRecordIsDeleted

S32 DbfRecordIsDeleted(sDBF *dbf);

功能： 返回当前记录是否打开的标志。

返回： 返回DBF_RECORD_NORMAL表示未删除；返回DBF_RECORD_DELETED表示已删除。小于0表示失败。

参数：

- 1) dbf 已打开数据库的指针。

示例：

```
S32 iRet;
    iRet = DbfRecordCount(&dbf);
if(iRet < 0) {...failed...}
else if(iRet == DBF_RECORD_NORMAL) {...not deleted...}
else if(iRet == DBF_RECORD_DELETED) {...deleted...}
```

DbfRecordLocate

S32 DbfRecordLocate(U16 field, S8 *pattern, U8 direction, U8 match, sDBF *dbf);

功能： 按规定的条件定位符合的记录。如果找到则改变当前记录号，否则不变。

返回： 返回DBF_OPER_OK表示定位成功，并将查找到的记录设为当前记录；小于0表示未查找到记录或其它错误。

参数：

- 1) field 匹配条件的字段号。0代表数据库的第1个字段。
- 2) pattern 要匹配的字符串。注意，此参数是大小写敏感的。
- 3) direction 查找的方向。当为DBF_LOCATE_UP时从当前记录向上查找；当为DBF_LOCATE_DOWN时从当前记录向下查找。当前记录也在被查找的范围内。
- 4) match 匹配的模式。当为DBF_LOCATE_MATCH_PART时为模糊匹配，只要字段中包含了pattern，那么就认为是匹配的；当为DBF_LOCATE_MATCH_ALL时为完全匹配，字段必须和pattern完全相同才认为是匹配的。
- 5) dbf 已打开数据库的指针。

示例：

```
S32 iRet;
S8 szField[256];
    //从当前记录向下查找第1个字段中含有 patern 的记录，模糊匹配
    iRet = DbfRecordLocate(0, "patern", DBF_LOCATE_DOWN, DBF_LOCATE_MATCH_PART, &dbf);
if(iRet < 0) {...failed...}
iRet = DbfFieldGet(1, szField, &dbf); //获取定位成功后的第2个字段值
```

DbfGetMatchCount

S32 DbfGetMatchCount(U16 field, S8 *pattern, U8 match, sDBF *dbf);

功能： 从第一条记录开始，按规定的条件统计符合的记录总数。完成后当前记录号不变。

返回：返回大于等于0表示统计成功，也即符合条件的记录总数；小于0表示未查找到记录或其它错误。

参数：

- 1) **field** 匹配条件的字段号。0代表数据库的第1个字段。
- 2) **pattern** 要匹配的字符串。注意，此参数是大小写敏感的。
- 3) **match** 匹配的模式。当为DBF_LOCATE_MATCH_PART时为模糊匹配，只要字段中包含了pattern，那么就认为是匹配的；当为DBF_LOCATE_MATCH_ALL时为完全匹配，字段必须和pattern完全相同才认为是匹配的。
- 4) **dbf** 已打开数据库的指针。

示例：

```
S32 iRet;
    //从第一条记录统计第1个字段中含有 pattern 的记录，模糊匹配
    iRet = DbfGetMatchCount(0, "pattern", DBF_LOCATE_DOWN, DBF_LOCATE_MATCH_PART, &dbf);
if(iRet < 0) {...failed...}
//iRet即为匹配成功的记录条数
```

DbfFieldSize

S32 DbfFieldSize(U16 field, sDBF *dbf);

功能：获取指定字段占用的空间大小，以字节为单位。

返回：返回大于等于0表示获取成功，也即指定字段占用的空间大小；小于0表示错误。

参数：

- 1) **field** 要获取尺寸的字段号。0代表数据库的第1个字段。
- 2) **dbf** 已打开数据库的指针。

示例：

```
S32 iRet;
    iRet = DbfFieldSize (0, &dbf);
if(iRet < 0) {...failed...}
//iRet即为第1个字段的长度
```

DbfFieldGet

S32 DbfFieldGet(U16 field, S8 *data, sDBF *dbf);

功能：获取当前记录指定字段的数据内容。

返回：返回DBF_OPER_OK表示获取成功；小于0表示错误。

参数：

- 1) **field** 要获取内容的字段号。0代表数据库的第1个字段。
- 2) **data** 存放字段内容的buffer。
- 3) **dbf** 已打开数据库的指针。

示例：

```
S32 iRet;  
S8 szField[256];  
    iRet = DbfFieldGet(0, szField , &dbf);  
if(iRet < 0) {...failed...}  
// szField即为第1个字段的内容，以0结尾
```

DbfFieldSet

S32 DbfFieldSet(U16 field, S8 *data, sDBF *dbf);

功能：设置当前记录指定字段的数据内容。

返回：返回DBF_OPER_OK表示设置成功；小于0表示错误。

参数：

- 1) **field** 要设置内容的字段号。0代表数据库的第1个字段。
- 2) **data** 存放字段内容的buffer。
- 3) **dbf** 已打开数据库的指针。

示例：

```
S32 iRet;  
S8 *pszField = "my fields content";  
    iRet = DbfFieldSet (0, pszField, &dbf);  
if(iRet < 0) {...failed...}  
// pszField已设置为第1个字段的内容
```

DbfFieldCount

S32 DbfFieldCount(sDBF *dbf);

功能：获取数据库的字段个数。

返回：返回大于等于0表示获取成功，也即指定数据库的字段个数；小于0表示错误。

参数：

- 1) **dbf** 已打开数据库的指针。

示例：

```
S32 iRet;  
    iRet = DbfFieldCount (&dbf);  
if(iRet < 0) {...failed...}  
// iRet即数据库的字段个数
```

DbfFieldInfo

S32 DbfFieldInfo(U16 field, sField *fieldinfo, sDBF *dbf);

功能：获取数据库中指定字段的信息。

返回：返回DBF_OPER_OK表示设置成功；小于0表示错误。

参数：

- 1) **field** 要获取信息的字段号。0代表数据库的第1个字段

- 2) **fieldinfo** 指定字段的信息。结构sField的说明请参考前面结构体定义中的sField结构。
- 3) **dbf** 已打开数据库的指针。

示例:

```
S32 iRet;
sField stFieldInfo;
    iRet = DbfFieldInfo (0, &stFieldInfo , &dbf);
if(iRet < 0) {...failed...}
// stFieldInfo即数据库的第0个字段的信息
```

DbfRecordDelete

`S32 DbfRecordDelete(sDBF *dbf);`

功能: 在数据库记录中设置删除标志。

返回: 返回DBF_OPER_OK表示设置成功；小于0表示错误。

参数:

- 1) **dbf** 已打开数据库的指针。

示例:

```
S32 iRet;
    DbfGotoRecord(record,&dbf) ;
    iRet = DbfRecordDelete(&dbf);
if(iRet < 0) {...failed...}
```

备注: 删除标志可以通过S32 DbfRecordIsDeleted(sDBF *dbf)函数来判断。

DbfRecordRestore

`S32 DbfRecordRestore (sDBF *dbf);`

功能: 取消被设置了删除标志的记录。

返回: 返回DBF_OPER_OK表示设置成功；小于0表示错误。

参数:

- 1) **dbf** 已打开数据库的指针。

示例:

```
S32 iRet;
    DbfGotoRecord(record,&dbf) ;
    iRet = DbfRecordRestore(0, &stFieldInfo , &dbf);
if(iRet < 0) {...failed...}
```

DbfCreate

`S32 DbfCreate(S8 *dbfname,U32 fieldcount,S8* fieldname[],U8 *fieldsize);`

功能: 创建给定的名字和字段信息的dbf文件。

返回: 返回DBF_OPER_OK表示创建成功；小于0表示错误。

参数:

- | | |
|----------------|---------------|
| 1) dbfname | 需要创建的dbf文件名称。 |
| 2) fieldcount | dbf内的字段数目 |
| 3) fieldname[] | 字段的名称，为指针数组 |
| 4) fieldsize | 字段的大小，为U8数组 |

示例：

```
#define FIELD_COUNT 2
S8* fieldname[FIELD_COUNT]={"N1","N2"};
U8 fieldsize[FIELD_COUNT]={9,9};

iRet =DbfCreate("chaobiao0.dbf", FIELD_COUNT,fieldname,fieldsize);
if(iRet < 0) {...failed...}
```

注意：创建的dbf文件为空表文件，没有记录，不能够直接对其进行读写操作，必须使用 [DbfRecordAppend](#)追加一个记录才可以进行记录和字段的读写操作。

DbfCopy

[S32 DbfCopy\(S8 *DisDbfName, S8 *SrcDbfName\);](#)

功能：复制[SrcDbfName](#)的字段信息到[DisDbfName](#)文件。如果[DisDbfName](#)存在，则自动删除该文件后进行复制。

返回：返回DBF_OPER_OK表示复制成功；小于0表示错误。

参数：

- | | |
|---------------|----------------|
| 1) SrcDbfName | 需要复制的源dbf文件名称。 |
| 2) DisDbfName | 需要写入的目的dbf文件名称 |

注意：

1. 使用该函数只是复制了一个表的字段信息，而不是复制了整个表。该函数功能相当于新建了一个和源表字段结构一样的新表，该表为空表，没有记录，不能够直接对其进行读写操作，必须使用 [DbfRecordAppend](#)追加一个记录才可以进行记录和字段的读写操作。

示例：

```
iRet =DbfCopy("chaobiao_copy.dbf","chaobiao.dbf");
if(iRet < 0) {...failed...}
```

DbfRecordErase

[S32 DbfRecordErase\(sDBF *dbf\);](#)

功能：物理删除当前记录，删除后不可恢复

返回：返回DBF_OPER_OK表示复制成功；小于0表示错误。

参数：

- | | |
|--------|------------|
| 1) dbf | 已经打开的数据库指针 |
|--------|------------|

注意：如果记录数比较多时，该函数比较耗时。如100条记录，删掉第一条，需要移动后面的99条。如果可能，可以考虑从后面删除，如删除第100条记录，不需要移动其他的数据，这样不会消耗太多的时间。

示例：

DbfFieldIndex

S32 DbfFieldIndex(S8 *fieldname, sDBF *dbf);

功能: 根据字段名称返回该字段的序号

返回: 小于0表示错误, 大于等于0表示改字段的序号

参数:

- 1) **fieldname** 字段的名称
- 2) **dbf** 已经打开的数据库指针

注意:

DbfRecordPack

S32 DbfRecordPack(sDBF *dbf);

功能: 物理删除已经被逻辑删除(使用DbfRecordDelete删除)的记录

返回: 0表示成功

参数:

- 1) **dbf** 已经打开的数据库指针

注意: 使用该函数需要进行数据移动, 比较耗时间.

DbfDelete

S32 DbfDelete (sDBF *dbf);

功能: 物理删除dbf文件, 删除后dbf文件不能使用

返回: 0表示成功

参数:

- 1) **dbf** 已经打开的数据库指针

注意: 使用该函数后, 该dbf文件被关闭, 其被删除, 不可恢复。

DbfRecordClear

S32 DbfRecordClear (sDBF *dbf);

功能: 物理删除所有的记录, 使该dbf文件成为一个没有记录的空dbf文件

返回: 0表示成功

参数:

- 1) **dbf** 已经打开的数据库指针

注意: 使用该函数会清除所有的记录, 并且不能恢复。

文件操作函数

使用该文件操作函数时，文件指针必须指向一个有效值，否则可能会造成系统访问内存失败而死机。

FileDelete

U32 FileDelete(sFILE *fp);

功能：删除文件

返回：成功删除则返回0，不成功则返回错误号

参数：

- 1) fp 当前已经赋值的有效文件指针

注意：

- 1. 使用该函数删除文件，文件不能通过通信软件恢复

FileChoiceRecycle

U32 FileChoiceRecycle(sFILE *fp, U32 choice)

功能：将文件放入回收站

返回：0表示成功执行，非0表示执行失败，或者没有处理

参数：

- 1) fp 当前已经赋值的有效文件指针。

2) choice 如果回收站已经有同名文件，则系统按choice指定方式处理，否则直接放入回收站。

choice=0表示删除自身文件，保留回收站文件

choice=1表示删除回收站所有同名文件后将本文件放入回收站

choice=2表示不处理回收站文件，直接将本文件放入回收站

choice>2不处理

注意：

- 1. 使用该函数删除文件，文件可以通过通信软件恢复，但该回收站的文件会占用系统存储空间。
- 2. 同一个文件，回收次数最多为7次，第8次回收则被删除，不能够通过通信软件恢复

FileClose

U32 FileClose(sFILE *fp);

功能：关闭一个已经打开的文件。

返回：0，表示正常关闭，非0表示关闭文件出现异常

参数：

- 1) fp 当前已经赋值的有效文件指针。

注意：

调用关闭文件函数后，不管文件是否正常关闭，fp的值都为0，如需要再次使用必须重新赋值

FileRead

U32 FileRead(U8 * buf, U32 size, sFILE *fp);

功能：从文件中读取指定大小数据。

返回：实际读取的字节数。

参数：

- 1) **buf** 存放文件内容的指针。
- 2) **size** 读取的字节数。
- 3) **fp** 当前已经赋值的有效文件指针

注意：

1. 当读取到文件末尾还不能满足size的大小时，返回实际读取到的字节数
2. 存放数据的缓冲区必须不能小于size的大小，否则将造成数据溢出而破坏其他数据，严重情况下会造成文件数据混乱，系统死机等情况

FileWrite

U32 FileWrite(U8* buf, U32 size, sFILE *fp);

功能：写数据到文件中。

返回：实际写入的字节数。

参数：

- 1) **buf** 要写入数据的指针。
- 2) **size** 要写入的字节数。
- 3) **fp** 当前已经赋值的有效文件指针。

注意：

当文件不是以追加方式写入时，写到文件末尾时退出函数，并返回实际写入文件的字节数

FileSeek

U32 FileSeek(sFILE *fp, S32 offset, U32 origin);

功能：定位当前文件指针。

返回：正确定位返回0，否则返回非0值。

参数：

- 1) **fp** 当前已经赋值的有效文件指针。
- 2) **offset** 偏移量，可为正数或者负数
- 3) **origin** 定位的开始位置。可以是以下三种之一(宏定义请参考头文件)。

SEEK_SET 从文件起始处开始定位，offset 只能为正数；

SEEK_CUR 从文件当前位置开始定位，offset可正可负；

SEEK_END从文件结束位置开始定位，offset只能为负；

注意：

文件写入或者读取后，文件指针将自动增加，故需要使用该函数定位需要写入或者读取文件内如

的位置

FileLength

U32 FileLength(sFILE *fp);

功能： 返回文件长度。

返回： 文件的长度（单位：字节）。

参数：

1) **fp** 当前已经赋值的有效文件指针。

FileTell

U32 FileTell(sFILE *fp);

功能： 返回当前文件指针相对于文件起始的位置（单位：字节）。

返回： 成功则返回当前位置相对于文件起始的位置的字节数。

参数：

1) **fp** 当前已经赋值的有效文件指针。

示例：

```
U8 buffer[40]={0};
FILE *p;
U32 Count;
p=FileOpen("Br.dbf", "r");
If(p != NULL) {
Count=FileTell(p);
Count+=32;
FileSeek(p, Count, SEEK_SET);
FileRead(buffer, 30, p);
FileSeek(p, Count, SEEK_END);
FileWrite(buffer, 30, p);
FileClose(p);
}
else
LcdPutstring("文件打开错误", 0xff);
```

FileOpen

sFILE *FileOpen(char *filename, char *mode);

功能： 打开(或新建)一个文件。

返回： 若正确打开文件则返回相应文件指针，否则返回 NULL。

参数：

1) **filename** 要打开(或新建)文件的文件名。

2) **mode** 文件打开方式串，该串中根据需要包含了r、w、a三种字符的组合。

r 以只读方式打开文件，文件指针定位在起始位置。

w 以写方式打开文件，文件指针定位在起始位置。

a 以追加方式打开文件，如果文件不存在则新建，文件指针定位在末尾。

注意：

使用该函数只能同时打开5个文件，如需要打开更多文件，请使用[FileOpenExpand](#)函数

FileOpenExpand

`sFILE * FileOpenExpand(char *filename, char *mode, sFILE *fp);`

功能：打开(或新建)一个文件，此函数与FileOpen的区别在于采用该函数将不受系统内部文件个数的限制。

返回：文件指针，正确打开时返回fp，否则返回0

参数：

- 1) **filename** 要打开(或新建)文件的文件名。
- 2) **mode** 文件打开方式串，该串中根据需要包含了r、w、a三种字符的组合。
r 以只读方式打开文件，文件指针定位在起始位置。
w 以写方式打开文件，文件指针定位在起始位置。
a 以追加方式打开文件，如果文件不存在则新建，文件指针定位在末尾。
- 3) **fp** 已经定义的sFILE类型变量地址

注意：

该函数需要一个sFILE类型的变量，fp为指向该变量的文件指针

示例：

```
sFILE file;
sFILE *fp;
fp= FileOpenExpand("test.txt", "wra", &file);
if (fp==NULL) {. . .}
else { . . . }
```

FileGetCreateTime

`void FileGetCreateTime(sFILE *fp, sRTC* createtime);`

功能：获取文件创建的时间

返回：无

参数：

- 1) **fp** 当前已经赋值的有效文件指针
- 2) **createtime** 返回的时间结构体

FileGetType

`U32 FileGetType(S8 *type, S8 *filename[], U32 count, U32 index);`

功能：获取指定类型的文件列表

返回：该类型文件总数

参数：

- 1) **type** 指定的文件类型，如“dbf”，使用“*. *”读取所有文件
- 2) **filename** 文件名指针数组，应为二维数组
- 3) **count** filename 中能够存放文件名的个数
- 4) **index** 将第几个开始的文件名填入 filename 中，用于循环显示，第一

次一般为 0，利用 index 能够实现多屏显示

注意：

filename要定义为指针数组，文件名长度固定为FILE_NAME_LENGTH，如：

```
int main(void) {
    int i;
    S8 name[8][ FILE_NAME_LENGTH];
    S8 *filename[8];
    for(i=0;i<8;i++)filename[i]=name[i]; //指针数组赋值
    FileGetType("dbf", (S8 **)filename, 8, 0);
}
```

输入法函数

InputSetParam

U32 InputSetParam(U32 type, U32 locked, U32 font);

功能：调用Input函数前设定默认的输入法，并可锁定输入法。

返回：操作的结果，0 操作成功，非0 操作失败。

参数：

- | | | |
|-----------|--|---|
| 1) type | 输入法索引，代表值如下(具体值请参考头文件中的定义)： | |
| | INPUT_TYPE_NUMBER 数字输入； | |
| | INPUT_TYPE_CHINESEPY 拼音输入； | |
| | INPUT_TYPE_CHINESEBH 笔画输入； | |
| | INPUT_TYPE_MIX 数字英文大小写混合输入； | |
| | INPUT_TYPE_CAPTITAL 数字英文大写； | |
| | INPUT_TYPE_LOWERCASE 数字英文小写； | |
| 2) locked | 是否锁定（锁定后按切换键不能改变输入法）。INPUT_UNLOCKED 表示不锁定；INPUT_LOCKED 代表锁定。 | 表 |
| 3) font | 字体大小，8，12，16 | |

Input

U32 Input(U32 x, U32 y, S8 *buffer, U32 size);

功能：输入数字、中文、英文、字符。

返回：最后的键值，一般是KEY_ENTER或是KEY_ESC。

参数：

- | | |
|-----------|----------------------|
| 1) x | 要显示输入内容的X轴坐标（0~159）。 |
| 2) y | 要显示输入内容的Y轴坐标（0~159）。 |
| 3) buffer | 存放输入内容的缓冲地址。 |
| 4) size | 缓冲区的大小。 |

注意：

使用该函数可以在不锁定输入法状态下进行输入法切换，按键盘右下角按键实现切换功能

示例：

```
U8 buffer[120]={0};  
Input(0,0,buffer,100); //在0,0的位置上输入100个字符。
```

InputBh

U32 InputBh(U32 x,U32 y,S8 *buffer,U32 size);

功能：笔画输入。

返回：最后的键值，一般是KEY_ENTER或是KEY_ESC。

参数：

- 1) **x** 要显示输入内容的X轴坐标（0～159）。
- 2) **y** 要显示输入内容的Y轴坐标（0～159）。
- 3) **buffer** 存放输入内容的缓冲地址。
- 4) **size** 缓冲区的大小

注意：

1. 该函数只能以笔画方式输入中文，标配只支持GB2312字库，可选GBK输入法
2. 输入时，系统将显示屏底部24行像素作为汉字选，故y的坐标不要超过136

InputMix

U32 InputMix(U32 x,U32 y,S8 *buffer,U32 size);

功能：数字英文大小写混合输入。

返回：最后的键值，一般是KEY_ENTER或是KEY_ESC。

参数：

- 1) **x** 要显示输入内容的X轴坐标（0～159）。
- 2) **y** 要显示输入内容的Y轴坐标（0～159）。
- 3) **buffer** 存放输入内容的缓冲地址。
- 4) **size** 缓冲区的大小

注意：

该函数能够输入除中文外的ASCII字符

InputYw

U32 InputYw(U32 x,U32 y,S8 *buffer,U32 size);

功能：英文输入。

返回：最后的键值，一般是KEY_ENTER或是KEY_ESC。

参数：

- 1) **x** 要显示输入内容的X轴坐标（0～159）。
- 2) **y** 要显示输入内容的Y轴坐标（0～159）。

- 3) **buffer** 存放输入内容的缓冲地址。
- 4) **size** 缓冲区的大小

注意：

该函数只能输入大小写的英文字母和符号

InputPy

U32 InputPy(U32 x,U32 y,S8 *buffer,U32 size);

功能：拼音输入。

返回：最后的键值，一般是KEY_ENTER或是KEY_ESC。

参数：

- 1) **x** 要显示输入内容的X轴坐标（0～159）。
- 2) **y** 要显示输入内容的Y轴坐标（0～159）。
- 3) **buffer** 存放输入内容的缓冲地址。
- 4) **size** 缓冲区的大小

注意：

- 1. 该函数只能以拼音方式输入中文，标配只支持GB2312字库，不支持GBK字库
- 2. 输入时，系统将显示屏底部24行像素作为汉字选，故y的坐标不要超过136

InputSz

U32 InputSz(U32 x,U32 y,S8 *buffer,U32 size);

功能：数字输入，符号键能够输入多种符号。

返回：最后的键值，一般是KEY_ENTER或是KEY_ESC。

参数：

- 1) **x** 要显示输入内容的X轴坐标（0～159）。
- 2) **y** 要显示输入内容的Y轴坐标（0～159）。
- 3) **buffer** 存放输入内容的缓冲地址。
- 4) **size** 缓冲区的大小

注意：

该函数只能输入数字和符号

InputPassword

U32 InputPassword(U32 x,U32 y,S8 *buffer,U32 size);

功能：数字输入，输入内容显示为'*'，具有输入密码的效果。

返回：最后的键值，一般是KEY_ENTER或是KEY_ESC。

参数:

- 1) **x** 要显示输入内容的X轴坐标（0～159）。
- 2) **y** 要显示输入内容的Y轴坐标（0～159）。
- 3) **buffer** 存放输入内容的缓冲地址。
- 4) **size** 缓冲区的大小

注意:

该函数与InputSz相同，只是不回显输入内容

InputSzn

U32 InputSzn (U32 x, U32 y, S8 *buffer, U32 size);

功能: 数字输入，符号键只能够输入点号和负号。

返回: 最后的键值，一般是KEY_ENTER或是KEY_ESC。

参数:

- 1) **x** 要显示输入内容的X轴坐标（0～159）。
- 2) **y** 要显示输入内容的Y轴坐标（0～159）。
- 3) **buffer** 存放输入内容的缓冲地址。
- 4) **size** 缓冲区的大小

注意:

该函数与InputSz不相同之处在符号键只能够输入点号和负号

键盘函数

KeyValue

U8 KeyValue(void);

功能: 获取最后一次按键的键值

返回: 最后的键值

参数: 无

KeyWait

U8 KeyWait(void);

功能: 等待用户按键。

返回: 按键键值，具体值请参考上表。

参数: 无。

注意:

该函数会使系统进入超低功耗停机状态，该模式下系统有些功能将失效，使用该函数能够最大程度延长电池使用时间，一般情况下选用该函数。如需使用到系统内部高级功能，而该功能不能够在该模式下工作，可以选用[KeySleepWait函数](#)

KeySleepWait

U8 KeySleepWait(void);

功能：等待用户按键。

返回：按键键值，具体值请参考上表。

参数：无。

注意：

该函数会使系统进入低功耗待机状态，该模式下系统所有功能有效，使用该函数能够延长电池使用时间，系统所有高级功能均可以使用

KeyTimeWait

U8 KeyTimeWait(U32 time);

功能：在规定的时间内等待用户按键，。

返回：按键键值，具体值请参考上表，超时返回0。

参数：

1) time 超时时间 ， 单位 mS（毫秒）

注意：

在time时间内没有按键按下，则返回0

KeyScan

U8 KeyScan(void);

功能：扫描键盘，并返回键值

返回：按键键值

参数：无

注意：

该函数不等待用户按键

KeySendValue

void KeySendValue(int key);

功能：向系统发送一个键值，以使等待安装的任务得以继续运行。通常用于界面刷新。

返回：无

参数：无

注意：

通常用于中断或者多任务工作时，一个后台任务向界面任务发送一个键值，以通知界面任务进行界面

刷新和更新处理的内容。

SetPowerOffFun

```
void SetPowerOffFun(pExFunction Funtion);
```

功能：设置关机时运行的函数

返回：无

参数：无返回值和输入参数函数的函数指针

注意：通常关机前处理一些事情，如回写参数变量，关闭外部设备等操作。

使用的函数原型为void Eample(void) { /*关机执行的代码*/ }

例子：

```
SetPowerOffFun(Eample);
```

SetPowerOnFun

```
void SetPowerOnFun(pExFunction Funtion);
```

功能：设置开机时运行的函数

返回：无

参数：无返回值和输入参数函数的函数指针

注意：通常用于重新初始一些硬件资源，以便于开机后硬件能够正常工作

使用的函数原型为void Eample(void) { /*关机执行的代码*/ }

例子：

```
SetPowerOnFun(Eample);
```

SetKeyFuntion

```
void SetKeyFuntion(pExFunction Funtion, S32 Key);
```

功能：设置F1, F2, Fn按下后默认调用的函数。F1, 系统默认为键盘背光功能, F2默认为液晶背光调节功能, Fn在EDIT, MULTIEDIT控件在获取焦点时, 按下默认调用中文输入功能。

返回：无

参数：1、无返回值和输入参数函数的函数指针

2、按键键值, F1为GUI_KEY_F1, F2为GUI_KEY_F2, Fn为GUI_KEY_Fn

注意：

使用的函数原型为void Eample(void) { /*要执行的代码, 没有代码时只取消原有功能*/ }

例子一：

```
void Eample(void) {}
```

```
SetKeyFuntion(Eample, GUI_KEY_F1); //取消原有背光开关功能
```

例子二：

```
SetKeyFuntion(NULL, GUI_KEY_F1); //恢复原有背光开关功能
```


显示函数

LcdClear

`void LcdClear(void);`

功能：清屏，清屏后屏幕无显示

返回：无。

参数：无

注意：

该函数为全屏清屏，要局部清屏，可以选用：[LcdSetArea](#)

LcdSetDot

`void LcdSetDot(U32 startx,U32 starty,U32 colour);`

功能：将指定位置的点设成指定颜色。

返回：无。

参数：

- 1) **startx** x轴方向位置（灰度屏0~159）（彩屏0~239）
- 2) **starty** y轴方向位置（灰度屏0~159）（彩屏0~239）
- 3) **colour** 指定的颜色（0~31）

LcdSetArea

`void LcdSetArea(U32 startx,U32 starty,U32 endx,U32 endy,U32 colour);`

功能：将指定区域显示成指定颜色，常用于局部清屏

返回：无。

参数：

- 1) **startx** 左上顶点X轴坐标（灰度屏0~159）（彩屏0~239）
- 2) **starty** 左上顶点Y轴坐标（灰度屏0~159）（彩屏0~319）
- 3) **endx** 右下顶点X轴坐标（灰度屏1~160）（彩屏1~240）
- 4) **endy** 右下顶点Y轴坐标（灰度屏1~160）（彩屏1~320）
- 5) **colour** 要设置的颜色（灰度屏0~31）（彩屏0~65535）

LcdSetColor

`void LcdSetColor(U32 color);`

功能：设置系统颜色

返回：无

参数：

1) color 显示字符的颜色，范围（灰度屏0～31）（彩屏0～65535）

LcdSaveArea

```
void LcdSaveArea (U32 startx,U32 starty,U32 endx,U32 endy,U8 *buffer,U32 zip);
```

功能：保存显示屏上某个区域的数据到缓冲区中。

返回：无。

参数：

- 1) startx 左上顶点X轴坐标（灰度屏0～159）（彩屏0～239）。
- 2) starty 左上顶点Y轴坐标（灰度屏0～159）（彩屏0～239）。
- 3) endx 右下顶点X轴坐标（灰度屏1～160）（彩屏1～240）
- 4) endy 右下顶点Y轴坐标（灰度屏1～160）（彩屏1～320）
- 5) zip 指明是否压缩数据，0表示不压缩，1表示压缩。压缩比为8。

注意：

1. 当您的显示方式是多级灰度时，如果使用压缩功能，得到的结果将是丢失灰度信息。
2. 由于不压缩数据将占用很多内存资源，建议使用压缩功能

Buffer计算方法：（endx－startx+6）×（endy－starty）。采用压缩时是前面结果除8取整加1；

LcdRestoreArea

```
void LcdRestoreArea(U32 startx,U32 starty,U32 endx,U32 endy,U8 *buffer,U32 zip);
```

功能：将缓冲区中的数据恢复到显示屏某个区域。

返回：无。

参数：

- 1) startx 左上顶点X轴坐标（灰度屏0～159）（彩屏0～239）
- 2) starty 左上顶点Y轴坐标（灰度屏0～159）（彩屏0～239）
- 3) endx 右下顶点X轴坐标（灰度屏1～160）（彩屏1～240）
- 4) endy 右下顶点Y轴坐标（灰度屏1～160）（彩屏1～320）
- 5) zip 指明缓冲中的数据是否是压缩数据，0表示不压缩，1表示压缩。

注意：

该函数一般与LcdSaveArea成对使用，当x，y坐标不与LcdSaveArea一致时，可以实现屏幕迁移显示，但zip必须一致，否则显示错乱。

LcdDrawLine

```
void LcdDrawLine(U32 startx,U32 starty,U32 endx,U32 endy);
```

功能：根据两点画一直线。

返回：无。

参数:

- 1) **startx** 起点X轴坐标（灰度屏0~159）（彩屏0~239）
- 2) **starty** 起点Y轴坐标（灰度屏0~159）（彩屏0~239）
- 3) **endx** 终点X轴坐标（灰度屏1~160）（彩屏1~240）
- 4) **endy** 终点Y轴坐标（灰度屏1~160）（彩屏1~320）

注意:

该函数可以画水平，垂直，斜线

LcdDrawRect

```
void LcdDrawRect(U32 startx,U32 starty,U32 endx,U32 endy);
```

功能: 根据两点画一矩形。

返回: 无。

参数:

- 1) **startx** 左上顶点X轴坐标（灰度屏0~159）（彩屏0~239）
- 2) **starty** 左上顶点Y轴坐标（灰度屏0~159）（彩屏0~239）
- 3) **endx** 右下顶点X轴坐标（灰度屏1~160）（彩屏1~240）
- 4) **endy** 右下顶点Y轴坐标（灰度屏1~160）（彩屏1~320）

LcdDisplayIcon

```
void LcdDisplayIcon(U32 startx,U32 starty,U8* icon,U32 mode,U32 type);
```

功能: 显示16级灰度的图标

返回: 无

参数:

- 1) **startx** 图标要显示的x轴位置，对应图标左上角x位置
- 2) **starty** 图标要显示的y轴位置，对应坐标左上角y位置
- 3) **icon** 存放图标数据的指针
- 4) **mode** 显示模式，0反显，1正常
- 5) **type** 图标的类型，分16×16，32×32，48×48类型，取值为16，32，48

注意:

- 1. 如果用户需要显示字节制作的图标，图标灰度为16级灰度，也就是一个字节存放两个像素点，扫描顺序为从左到右，从上至下
- 2. 如果需要显示系统自带图标，图标地址由 [LcdGetIconAddr](#) 函数获取

LcdPutChar

```
void LcdPutChar(U32 data,U32 mode,U32 font);
```

功能： 在当前位置显示一个字符。

返回： 无。

参数：

- 1) `pdata` 要显示的字符串。
- 2) `mode` 显示模式（0反显，1正常，2嵌入并反显，3嵌入并正常显示）。
- 3) `font` 字体大小，可选16、12、8

注意：

该函数只能显示一个ASCII字符

LcdPutNChar

```
void LcdPutNChar(S8 *pdata,U32 count,U32 mode,U32 font);
```

功能： 在当前位置显示不多于count个字符。

返回： 无。

参数：

- 1) `pdata` 要显示的字符串。
- 2) `count` 要显示的字符个数。
- 3) `mode` 显示模式（0反显，1正常，2嵌入并反显，3嵌入并正常显示）。
- 4) `font` 字体大小，可选16、12、8

注意：

如果显示的数据长度小于给定的长度，将只显示数据长度的字节数

LcdPutString

```
void LcdPutString(S8 *pdata,U32 mode,U32 font);
```

功能： 在当前位置显示字符串。

返回： 无。

参数：

- 1) `pdata` 要显示的字符串。
- 2) `mode` 显示模式（0反显，1正常，2嵌入并反显，3嵌入并正常显示）。
- 3) `font` 字体大小，可选16、12、8

注意：

该函数只能显示不多于256个字符的数据，超过部分将被截取

LcdPrintf

```
void LcdPrintf(U32 mode,U32 font,char *fmt,...);
```

功能： 可变参数个数，可将各种数据类型的数据格式化后输出到显示屏，输出字符大小为12×12的汉字或是12×6的ascii字符。

返回： 无。

参数：

- 1) `mode` 显示模式（0 反显，1 正常，2 嵌入并反显，3 嵌入并正常显示）

2) **font** 字体大小，可选 16、12、8

2) **fmt** 格式化串。用于控制转换后串的格式。包括

%d 有符号整型

%u 无符号整型

%x 无符号十六进制数

%f 带符号浮点数

%e 带符号浮点数(科学计数法表示)

%c 字符

%s 字符串

%p 指针

注意：

该函数只能显示不多于256个字符的数据，超过部分将被截取

LcdMoveto

```
void LcdMoveto(U32 startx,U32 starty);
```

功能：移动光标位置。

返回：无。

参数：

1) **startx** X轴坐标（灰度屏0~159）（彩屏0~239）。

2) **starty** Y轴坐标（灰度屏0~159）（彩屏0~319）。

注意：

通常情况下，需要在指定的位置输出信息就要调用该函数，否则显示函数将接着上一输出位置继续显示

LcdSetCursor

```
void LcdSetCursor(U32 status,U32 width,U32 height,U32 blink);
```

功能：设置光标。

返回：无。

参数：

1) **status** 光标开关。1表示使用光标，0表示停止光标

2) **width** 光标宽度。

3) **height** 光标高度。

4) **blink** 闪烁频率。

注意：

光标最大是16*8的大小，闪烁频率目前固定为一秒，这个函数起打开与关闭光标的功能。

例如：

```
LcdSetCursor(1, 8, 16, 0);
```

```
KeyWait();
```

LcdSetCursor(0, 8, 16, 0);

LcdDrawArcAngle

void LcdDrawArcAngle(S32 startx, S32 starty, S32 stangle, S32 endangle, S32 radius, S32 mode);

功能:以半径方式画圆弧

返回: 无

参数:

- 1) startx 圆弧起点 x 轴坐标, (灰度屏 0~159) (彩屏 0~239)
- 2) starty 圆弧起点 y 轴坐标, (灰度屏 0~159) (彩屏 0~319)
- 3) stangle 圆弧起始角度, 0~360
- 4) endangle 圆弧终点角度, 0~360
- 5) radius 圆弧半径, 最大值 1.414×160
- 6) mode 圆弧方向, 0表示顺时针圆弧, 1表示逆时针圆弧

注意:

当参数标超过规定数值时将无法显示, 圆弧参数之间相互制约, 注意选择

LcdDrawArcCenter

void LcdDrawArcCenter(S32 centrex, S32 centrey, S32 startx, S32 starty, S32 endx, S32 endy, U32 mode);

功能: 以圆心方式画圆弧

返回: 无

- 1) centrex 中心点 x 坐标, (灰度屏 0~159) (彩屏 0~239)
- 2) centrey 中心点 y 坐标, (灰度屏 0~159) (彩屏 0~319)
- 3) startx 起点 x 坐标, (灰度屏 0~159) (彩屏 0~239)
- 4) starty 起点 y 坐标, (灰度屏 0~159) (彩屏 0~319)
- 5) endx 终点 x 坐标, (灰度屏 0~159) (彩屏 0~239)
- 6) endy 终点 y 坐标, (灰度屏 0~159) (彩屏 0~319)
- 7) mode 圆弧方向, 0表示顺时针圆弧, 1表示逆时针圆弧

注意:

当参数标超过规定数值时将无法显示, 圆弧参数之间相互制约, 注意选择

LcdGetIconAddr

U8 *LcdGetIconAddr(U32 index, U32 type);

功能：获取指定的某个系统图标地址

返回：指定系统某个图标的地址

参数：

1) **index** 系统内部图标序号，参考常量定义里面的图标定义

2) **type** 图标类型，可用值 16，32，48

LcdGetX

U32 LcdGetX(void);

功能：获取当前系统X坐标

返回：当前系统X轴坐标

参数：无

LcdGetY

U32 LcdGetY(void);

功能：获取当前系统Y坐标

返回：当前系统Y轴坐标

参数：无

LcdSetBgColor

void LcdSetBgColor(U32 color);

功能：设置背景颜色

返回：无

参数：颜色(对应彩屏机器, 为16位色)

RGB888ToRGB565

S32 RGB888ToRGB565(S32 dst);

功能：将24位色转为16位色

返回：16位色

参数：24位色(选取颜色可以查看<RGB色彩对照表>)

LcdGetColor

U32 LcdGetColor(U32 startx, U32 starty);

功能：获取屏幕上指定点的色彩

返回：16位色

参数：屏幕X/Y坐标

菜单函数

MenuDisplay

S32 MenuDisplay(sMENU *menuparam, S8 *menu[]);

功能：显示菜单。

返回：选中菜单的索引（0~menuparam->amount）。

参数：

- 1) **menuparam** 要显示的菜单控制结构体，具体说明请参考结构pmStruct。
- 2) **menu** 要显示输入内容的Y轴坐标（0~159）。

注意：

菜单的显示位置与mode和位置参数相关，如果mode模式不是0，**menuparam** 的位置参数将会被修改。

示例：

```
const S8 * MainMenu[]={
"1. 运行指定程序",
"2. 运行选定程序",
"3. 察看系统信息",
"4. 通讯管理设置",
"5. 进入文件管理",
"6. 进入系统设置",
"7. 察看万年历",
"8. 使用计算器",
"9. 帮助"};
U8 menuchoice;
pmStruct menuparam;
menuparam.startx=5;
menuparam.starty=5;
menuparam.endx=100;
menuparam.endy=80;
menuparam.mode=1; //0: fixed mode 1: center mode 2: expand mode
                  //4: main mode 5: submode
menuparam.count=6;
menuparam.border=1;
menuparam.amount=9;
menuparam.hotkey="123456789";
menuparam.current=0;
menuparam.style=16;
cls();
menuchoice=MenuDisplay(&menuparam, (S8 *)MainMenu);
```

MenuTextMain

S32 MenuTextMain(sMENU *menuparam, const S8 *menu[], sMENU *submenuparam, const S8 *submenu[], S8 *subamount);

功能：建立主菜单和子菜单二级菜单

返回：主菜单和子菜单的结果，当只选中主菜单没有选中子菜单，子菜单返回结果为-1；退出时返回-1；

参数:

- 1) **Menuparam** 要显示的主菜单控制结构体，具体说明请参考结构sMENU。
- 2) **menu** 主菜单 指针数组
- 3) **Submenuparam** 要显示的子菜单控制结构体，具体说明请参考结构sMENU。
- 4) **submenu** 该数组包含各个子菜单的指针数组
- 5) **subamount** 该数组指示每个子菜单包含的菜单数目

注意:

使用该函数能够实现二级菜单显示功能，此时需要配置主菜单参数中**menuparam.mode=4**;而从菜单参数中**submenuparam.mode=5**;

例子:

MenuDisplayCenter

S32 MenuDisplayCenter(S8* menu[], S8 *keys, U8 menu_count, U8 count, U8 border);

功能: 在屏幕的中间显示菜单。

返回: 选中菜单的索引 (0~menu_count) 。

参数:

- 1) **menu[]** 要显示的菜单指针数组
- 2) **keys** 热键字符串指针
- 3) **menu_count** 总的菜单数目。
- 4) **count** 单页可显示的菜单数目
- 5) **border** 是否显示边框

注意:

该函数只能显示在屏幕中央

MenuDisplayIcon

S32 MenuDisplayIcon(S8* menu[], U8* keys, int MenuCount);

功能: 显示图形菜单。

返回: 选中菜单的索引 (0~MenuCount) 。

参数:

- 1) **menu** 每项菜单的文字说明的指针数组。
- 2) **keys** 每项菜单使用的图标索引，其在数组中的位置与menu数组中的位置要对应。(大小为0——40) 既抄表机中共包含40个图标。用户可以根据自己的喜好自己选择图标。
- 3) **MenuCount** 菜单数目。

注意:

使用该函数能够实现图形化显示

示例:

```
S8 menuchoice;  
U8 index[7]={9, 10, 11, 12, 13, 14, };//用到抄表机中第9、10、11等6个图标  
LcdClear();  
menuchoice=MenuDisplayIcon(CommunicateMenu, index, 6);
```

MenuGraphMain

```
S32 MenuGraphMain(const S8* menu[], const U8* iconindex, int MenuCount, const S8* subiconindex[], U8  
* subcount);
```

功能: 显示图形菜单。

返回: 选中菜单的索引 (0~MenuCount)。

参数:

- | | |
|-----------------|---|
| 1) menu | 主菜单每项菜单的文字说明的指针数组。 |
| 2) iconindex | 主菜单每项菜单使用的图标索引，其在数组中的位置与menu数组中的位置要对应。（大小为0---40）既抄表机中共包含40个图标。用户可以根据自己的喜好自己选择图标。 |
| 3) MenuCount | 主菜单数目 |
| 4) subiconindex | 子菜单每项菜单使用的图标索引，其在数组中的位置与menu数组中的位置要对应。（大小为0---40）既抄表机中共包含40个图标。用户可以根据自己的喜好自己选择图标。 |
| 5) subcount | 子菜单数目 |

注意:

使用该函数能够实现图形菜单二级显示功能

示例:

延时函数

DelayMicrosecond

```
void DelayMicrosecond(U32 microsecond );
```

功能: 延时指定时间

返回: 无

参数:

- | | |
|----------------|----------------|
| 1) microsecond | 延时时间，单位 us（微秒） |
|----------------|----------------|

注意:

该函数最大延时时间30000微秒，如需延时更长时间，请使用DelayMillisecond

DelayMillisecond

`void DelayMillisecond(U32 millisecond);`

功能：延时指定时间，

返回：无

参数：

- 1) `millisecond` 延时时间，单位 ms（毫秒）

注意：

该函数最大延时时间30000毫秒，如需延时更长时间，请使用循环调用

串口函数

UartDeviceInit

`U32 UartDeviceInit(U32 mode, sUART * pdevice);`

功能：设置串行设备的工作模式，根据设备是否提供多种工作模式而选用

返回：0表示设置成功

参数：

- 1) `mode` 工作模式，一般有命令模式和数据模式
- 2) `pdevice` 串口类型指针

注意：

通常情况下不需要使用该函数，有些设备需要通过串口发送命令来初始化，此时则可调用该函数，是设备进入命令模式而非工作模式

UartOpen

`sUART * UartOpen(U32 baud, U32 mode, U32 device);`

功能：打开指定串行设备

返回：串口类型指针

参数：

- 1) `baud` 设备工作的波特率
- 2) `mode` 串口工作模式（数据位，停止位等）
- 3) `device` 设备索引

UartClose

`U32 UartClose(sUART * pdevice);`

功能：关闭指定串行设备

返回：0

参数：

- 1) `pdevice` 串口类型指针，当指针为NULL时关闭所有设备

UartRead

```
U32 UartRead(U8 *data,U32 size,U32 timeout,sUART * pdevice);
```

功能：读取指定长度数据

返回：实际接收到的数据大小

参数：

- 1) `data` 存放数据的指针
- 2) `size` 要读取的数据大小
- 3) `timeout` 超时退出时间，单位毫秒(mS)，0表示一直接收到固定大小才退出
- 4) `pdevice` 串口类型指针

UartWrite

```
U32 UartWrite(U8 *data,U32 size,U32 timeout,sUART * pdevice);
```

功能：发送指定长度数据

返回：实际发送的数据大小

参数：

- 1) `data` 读取数据的指针
- 2) `size` 要写入的数据大小
- 3) `timeout` 超时退出时间，单位毫秒(mS)，0表示一直接收到固定大小才退出
- 4) `pdevice` 串口类型指针

UartRxByte

```
U32 UartRxByte(U8 *data,U32 timeout,sUART *pdevice);
```

功能：接收一个字节的的数据

返回：实际接收的数据大小

参数：

- 1) `data` 读取数据的指针
- 2) `timeout` 超时退出时间，单位毫秒(mS)，0表示一直接收到固定大小才退出
- 3) `pdevice` 串口类型指针

UartTxByte

U32 UartTxByte(U8 data,U32 timeout,sUART *pdevice);

功能： 发送一个字节的数据

返回： 实际发送的数据大小

参数：

- 1) data 要发送的数据
- 2) timeout 超时退出时间， 单位毫秒(mS)， 0表示发送完数据后才能退出
- 3) pdevice 串口类型指针

UartRxString

U32 UartRxString(char *string ,U32 size,U32 Ftimeout,U32 Btimeout,sUART * pdevice);

功能： 接收小于等于指定长度的数据，可以按退出键和删除键结束接收状态，具体键值可以用KeyValue() 这个函数获取

返回： 无。

参数：

- 1) sting 接收到字符串存放缓冲
- 2) size 最大接收长度
- 3) Ftimeout 第一个字节的等待超时时间 ， 单位毫秒(mS)
- 4) Btimeout 后续字节的接收超时时间 ， 单位毫秒(mS)
- 5) pdevice 串口类型指针

UartTxString

U32 UartTxString(char *string,U32 size,U32 Btimewait,U32 timeout,sUART *pdevice);

功能： 发送小于等于指定长度的数据，可以按退出键和删除键结束接收状态，具体键值可以用KeyValue() 这个函数获取

返回： 实际发送了数据的大小。

参数：

- 1) sting 要发送的数据地址
- 2) size 最大发送长度
- 3) Btimewait 字节间的延时间隔时间 ， 单位毫秒(mS)
- 4) timeout 发送所有数据总的超时时间 ， 单位毫秒(mS)
- 5) pdevice 串口类型指针

UartPrintf

```
void UartPrintf(sUART *pdevice, char *fmt, ...);
```

功能： 可将各种数据类型的数据格式化后输出到串口。

返回： 无。

参数：

1) **pdevice** 串口类型指针

2) **fmt**

format 格式化串。用于控制转换后串的格式。

包括：

%d 有符号整型；

%u 无符号整型；

%x 无符号十六进制数；

%f 带符号浮点数；

%e 带符号浮点数(科学计数法表示) ；

%c 字符；

%s 字符串；

%p 指针；

注意：

同**LcdPrintf**函数一样，只是打印的位置不是屏幕而是串口。

变量函数

ParamRead

```
void * ParamRead(const S8* name);
```

功能： 读取指定的参数值。

返回： 指向参数值的字符串，以0结尾。如果参数为空，返回NULL，即(void *)0。

参数：

1) **name** 指明要获取值的参数名称。

示例：

```
char *pname;  
LcdMoveto(8, 28);  
LcdPutString("抄表员姓名", 0xff);  
pname=ParamRead("UserName");  
if (pname!=NULL) LcdPutString(pname, 0xff);
```

读取出厂条码：ParamRead("出厂条码");

ParamDelete

```
void ParamDelete(const S8* name);
```

功能： 删除指定的参数。

返回： 删除的结果，0 删除成功，非0 删除失败。

参数:

- 1) **name** 指明要删除的参数名称。

示例:

```
LcdMoveto(8, 28);  
LcdPutString("删除抄表员姓名", 0xff);  
Flag= ParamDelete("UserName");  
if (Flag==0) LcdPutString("删除成功", 0xff);  
else LcdPutString("删除失败", 0xff);
```

ParamWrite

`S32 ParamWrite(const S8* name, const void * pdata, U32 len);`

功能: 写入参数

返回: 写入的结果, 0 写入成功, 非0 写入失败。

参数:

- 1) **name** 指明要写的参数名称。
- 2) **pdata** 指向参数内容的指针
- 3) **len** 参数内容长度。

示例:

```
ParamWrite("SystemDisplayMode", "GraphMode", 10);
```

ParamEarse

`void ParamEarse(void);`

功能: 删除所有的用户变量

返回: 无

参数: 无

ParamSetWord

`void ParamSetWord(U32 index, S32 data);`

功能: 保存一个字的数据

返回: 无

参数:

- 1) **index** 存放的位置, 0~1023
- 2) **data** 要存放的32位数据

注意:

当 index 的取值范围超过了规定范围, 该数据不被存放。数据存放在内存中, 掉电保留

示例:

```
int a=10;  
ParamSetWord(0, a);
```

ParamGetWord

S32 ParamGetWord(U32 index);

功能： 读取一个字的数据

返回： 存放在该位置的一个32位数据

参数：

1) **index** 存放的位置，0~1023

注意：

当 index 的取值范围超过了规定范围，该数据返回 0

示例：

```
int a;  
a=ParamSetWord(0);
```

时间函数

RtcSetTime

void RtcSetTime(U8 Hour, U8 Minute, U8 Seconds);

功能： 设置系统时间

返回： 无。

参数：

1) **Hour** 小时，24小时值，十六进制表示，0~23。

2) **Minute** 分钟，0~59。

3) **Seconds** 秒，0~59。

示例：

```
RtcSetTime (13, 23, 10);
```

RtcSetAlarm

void RtcSetAlarm(U8 Hour, U8 Minute, U8 Seconds);

功能： 设置系统闹钟

返回： 无。

参数：

1) **Hour** 小时，24小时值，十六进制表示，0~23。

2) **Minute** 分钟，0~59。

3) **Seconds** 秒，0~59。

示例：

```
RtcSetAlarm (13, 23, 10);
```


RtcSetDate

```
void RtcSetDate( U16 Year, U8 Month, U8 Day);
```

功能： 设置系统日期

返回： 无。

参数：

- 1) Year 年，默认从2000年开始，取值0～99
- 2) Month 月，1～12
- 3) Day 日，1～31

示例：

```
RtcSetDate (8, 8, 8);
```

RtcGetDate

```
void RtcGetDate(U8 *year, U8 *month, U8 *day);
```

功能： 获取系统日期

返回： 无。

参数：

- 1) Year 年，默认从2000年开始，取值0～99
- 2) Month 月，1～12
- 3) Day 日，1～31

RtcGetWeek

```
void RtcGetWeek(U8 *week);
```

功能： 获取系统日期对应星期

返回： 无。

参数：

- 1) week
返回值，0表示周日，6表示周六

RtcGetTime

```
void RtcGetTime(U8 *hour, U8 *minute, U8 *second);
```

功能： 获取系统时间

返回： 无。

参数：

- 1) hour 小时，24小时值，十六进制表示，0～23。
- 2) minute 分钟，0～59。
- 3) seconds 秒，0～59。

RtcGetString

```
void RtcGetString(S8* pd);
```

功能：获取日期字符串

返回：无。

参数：

- 1) pd 返回格式 “2008\08\08”

RtcSetDateString

```
void RtcSetDateString(S8 *pd);
```

功能：获取时间字符串

返回：无。

参数：

- 1) pt 返回格式 “13:08:08”

RtcGetTimeString

```
void RtcGetTimeString(S8* pt);
```

功能：设置日期

返回：无。

参数：

- 1) pd 格式 “2008\08\08”

RtcSetTimeString

```
void RtcSetTimeString(S8* pt);
```

功能：设置时间

返回：无。

参数：

- 1) pt 格式 “13:08:08”

RtcDisplayOn

```
void RtcDisplayOn(U8 x,U8 y,U8 start,U8 end,S8 mode,U8 fonttype);
```

功能：设置定时刷新的时间显示

返回：无

参数：

- 1) x 时间显示的起点x轴坐标, (0~159)
- 2) y 时间显示的起点y轴坐标, (0~159)
- 3) start 时间显示字符串的起始位置 (0~19)

- 4) **end** 时间显示的终点位置(0~19)
- 5) **mode** 文字显示模式(0~3)
- 6) **fonttype** 文字显示的字体大小(8, 12, 16)

注意：时间格式为“2008\08\08 13:12:11”

示例：

以16字体反显模式显示年份

```
RtcDisplayOn(0, 0, 0, 3, 0, 16);
```

RtcDisplayOff

```
void RtcDisplayOff(void);
```

功能：关闭时间显示

返回：无

参数：无

注意：关闭时间显示后屏幕时间将不会更新

C 库函数

注意：如果使用到C函数库里面的 puts、printf、sprintf、atoi、atof、atol 函数，请使用下面函数替换，其他库函数，字符串类，如 strlen，memset，数学类，如 sin，cos 等可以直接使用，如果出现使用了库函数而不能编译出结果，可以联系本公司解决问题。

Lib_puts

```
int Lib_puts(const char * s);
```

功能：可将各种数据类型的数据输出到屏幕。

返回：无。

参数：

- 1) **s** 指向给定字符串的指针

注意：

该函数和**LcdPutString**区别是不能改变字体颜色和显示模式，只能是以16号字体显示

Lib_printf

```
int Lib_printf(const char * format, ...);
```

功能：可将各种数据类型的数据格式化后输出到屏幕。

返回：无。

参数：

1) **format** 格式化串。用于控制转换后串的格式。包括：

- %d 有符号整型；
- %u 无符号整型；
- %x 无符号十六进制数；
- %f 带符号浮点数；
- %e 带符号浮点数(科学计数法表示) ；
- %c 字符；
- %s 字符串；
- %p 指针；

注意：

该函数和**LcdPrintf**区别是不能改变字体颜色和显示模式，只能是以16号字体显示

Lib_sprintf

```
int Lib_sprintf(char * s, const char * format, ...);
```

功能： 可将各种数据类型的数据格式化后输出到 s 字符串中。

返回： 无。

参数：

1) s 指向给定字符串的指针

2) **format** 格式化串。用于控制转换后串的格式。包括：

- %d 有符号整型；
- %u 无符号整型；
- %x 无符号十六进制数；
- %f 带符号浮点数；
- %e 带符号浮点数(科学计数法表示) ；
- %c 字符；
- %s 字符串；
- %p 指针；

注意：

同printf函数一样，只是打印的位置不是给定的字符串。

Lib_atof

```
double Lib_atof(const char * nptr);
```

功能： 将字符串转换成浮点数

返回： 浮点数。

参数：

1) nptr 指向给定字符串的指针

Lib_atoi

```
int Lib_atoi(const char * nptr);
```

功能： 将字符串转换成整数

返回： 浮点数。

参数：

1) nptr 指向给定字符串的指针

Lib_atol

```
long int Lib_atol(const char * nptr);
```

功能：将字符串转换成长整数

返回：浮点数。

参数：

1) nptr 指向给定字符串的指针

Lib_srand

```
void Lib_srand(unsigned int seed);
```

srand 函数是随机数发生器的初始化函数。

用法：它需要提供一个种子，这个种子会对应一个随机数，如果使用相同的种子后面的 **rand()**函数会出现一样的随机数。如：**srand(1)**；直接使用 **1** 来初始化种子。

Lib_rand

```
int Lib_rand(void);
```

功 能：伪随机数发生器

所属库：stdlib.h

用 法：

需要先调用 **srand** 初始化，一般用当前日历时间初始化随机数种子，这样每次执行代码都可以产生不同的随机数。

Lib_calloc

```
void * Lib_calloc(unsigned int nmemb, unsigned int size);
```

功 能：在内存的动态存储区中分配n个长度为size的连续空间，函数返回一个指向分配起始地址的指针；如果分配不成功，返回NULL。

跟malloc的区别：

calloc在动态分配完内存后，自动初始化该内存空间为零，而**malloc**不初始化，里边数据是随机的垃圾数据。

用 法：void *calloc(unsigned n,unsigned size);

Lib_free

```
void Lib_free(void * ptr);
```

功 能：释放 ptr 指向的存储空间。被释放的空间通常被送入可用存储区池，以后可在调用 **malloc**、**realloc** 以及 **realloc** 函数来再分配。

Lib_malloc

```
void * Lib_malloc(unsigned int size);
```

功能：分配长度为num_bytes字节的内存块

返回值：如果分配成功则返回指向被分配内存的指针(此存储区中的初始值不确定)，否则返回空指针NULL。当

内存不再使用时，应使用free()函数将内存块释放。函数返回的指针一定要适当对齐，使其可以用于任何数据对象。

Lib_realloc

```
void * Lib_realloc(void * ptr, unsigned int size);
```

功能：先判断当前的指针是否有足够的连续空间，如果有，扩大 mem_address 指向的地址，并且将 mem_address 返回，如果空间不够，先按照 newsize 指定的大小分配空间，将原有数据从头到尾拷贝到新分配的内存区域，而后释放原来 mem_address 所指内存区域（注意：原来指针是自动释放，不需要使用 free），同时返回新分配的内存区域的首地址。即重新分配存储器块的地址。

返回值：如果重新分配成功则返回指向被分配内存的指针，否则返回空指针 NULL。

注意：这里原始内存中的数据还是保持不变的。当内存不再使用时，应使用 free() 函数将内存块释放。

调试函数

Bell

```
void Bell(U32 time,U32 tone);
```

功能：输出指定时间的音调

返回：无。

参数：

1) time 声音时长，单位毫秒(mS)

2) tone 声音音调

注意：

声音音调分7级，由1~7表示，对应do、re、mi、fa、so、la、si，0表示系统默认音调

LedLeft

```
void LedLeft(S32 onoff);
```

功能：点亮或者关闭左侧指示灯

返回：无。

参数：

1) onoff 1 点亮，0 关闭

LedMiddle

```
void LedMiddle(S32 onoff);
```

功能：点亮或者关闭中间指示灯

返回：无。

参数：

1) onoff 1 点亮，0 关闭

BackLightLcd

`void BackLightLcd (S32 onoff);`

功能： 点亮或者关闭液晶显示屏背光，使用于黑白屏机器（CL928）

返回： 无。

参数：

1) onoff 1 点亮, 0 关闭

BackLightLed

`void BackLightLed (S32 onoff);`

功能： 点亮或者关闭液晶显示屏背光，使用于彩屏机器（CL980、CL998）

返回： 无。

参数：

1) onoff 1 点亮, 0 关闭

BackLightKey

`void BackLightKey (S32 onoff);`

功能： 点亮或者关闭键盘背光

返回： 无。

参数：

1) onoff 1 点亮, 0 关闭

DetectVoltage

`U32 DetectVoltage(void);`

功能： 获取当前电池电压状态

返回： 电池电压状态等级，0~10。

参数： 无

DetectTemperature

`U32 DetectTemperature(void);`

功能： 获取机器内部温度

返回： 机器内部温度单位度

BatBarDisplayON

`void BatBarDisplayON(S32 x, S32 y, S32 periods);`

功能： 定时显示电池状态栏

返回： 无。

参数：

1) x X轴方向坐标

- 2) **y** Y 轴方向坐标
- 3) **preiolds** 周期, 单位秒, 不能超过 255 秒

注意:

状态栏的大小为 1 0 * 3 8 像素

BatBarDisplayOff

`void BatBarDisplayOff(void);`

功能: 关闭电池状态栏显示

返回: 无。

PowerManage

`void PowerManage(U32 powerbit,U8 state);`

功能: 功耗管理

返回: 无。

参数:

- 1) **powerbit** 使用的标志位, 取值范围 (1~0x80) 共 8 位, 可以用于八个任务或事件
- 2) **state** 标志位清 0 或者置 1

注意:

通常系统会处于停机模式, 以减少功耗, 如果程序中使用了 CPU 的内部资源, 而该资源在停机模式下无法使用, 如串口中断, 则需要将某一个标志位置 1

例如: `PowerManage(1, 1);` //申请系统不进入停机模式, 可以响应串口之类的中断

`PowerManage(1, 0);` //本任务可以停机使用, 释放申请的电源资源

ProgramLock

`void ProgramLock(char *ProgKey, char *CpuID);`

功能: 加密程序, 用户利用该函数控制程序可以在某一个或者某一类机器上运行, 如果不符合要求, 程序将不执行, 系统将自动重启。

返回: 无。

参数:

- 1) **ProgKey** 程序密码, 可以通过通信软件设置在机器里面, 这样程序可以运行在一批相同程序密码的机器。为 NULL 空时不检测。密码最多 30 个字符。
- 2) **CpuID** 机器硬件唯一编码, 如果不为 NULL, 则必须运行在指定的机器。不为 NULL 且参数不对时, 屏幕会给出机器编码, 填入后重新编译生成程序。

注意:

机器内部程序密码可以通过通信软件进行设置。程序密码和机器编码可以同时使用, 也可以分别使用。机器的硬件编码, 和出厂编码是不同。

例如: `ProgramLock("12345", NULL);` //在设置了程序密码为 12345 的机器上可以运行该程序

`ProgramLock("12345", "672384503245");` //在设置了程序密码为 12345 的机器上可以运行该程序, 并且只能够运行在机器硬件编码为 672384503245 的机器上。

UsbSendStringToPC

```
void Lcd2Bmp24Bit(void);
```

功能：以当前时间为文件名，保存当前显示屏的内容

返回：无。

参数：无

注意：

该函数常用于截取程序界面，便于编写使用说明书

USB 通信函数

UsbSendStringToPC

```
void UsbSendStringToPC(S8 *string);
```

功能：将字符串发送到电脑端

返回：无。

参数：

1) **string** 指向给定字符串的指针

注意：

只有手持机连接了电脑才能实现该功能

UsbSendKeyToPC

```
void UsbSendKeyToPC(U8 key,U8 control);
```

功能：将按键键值发送到电脑端

返回：无。

参数：

1) **key** 按键键值, 如 a 或者 A 键对应为十六进制的 0x04

2) **control** alt 键、shift 键、ctrl 键的状态，对应位为 1 表示该键按下

注意：

只有手持机连接了电脑才能实现该功能，键盘上对应的键值见附录

UsbSendAsciiToPC

```
void UsbSendAsciiToPC(S8 ascii);
```

功能：将ASCII字符发送到电脑端

返回：无。

参数：

1) **ascii** ASCII 字符，如 ‘1’ 对应十六进制的 0x31

注意：

只有手持机连接了电脑才能实现该功能

UsbConnectToPC

`void UsbConnectToPC(U32 type);`

功能：将手持机与电脑连接

返回：无。

参数：

1) **type** 连接类型，

该值为 0，手持机表现为将专用设备，使用本公司的通信协议。一般不使用，而直接使用 UsbMain 函数。

该值为 1，表示以键盘方式连接连接到电脑，此时可以使用 UsbSendStringToPC，UsbSendKeyToPC，UsbSendAsciiToPC 函数将内容以按键输入的方式发到电脑，简单而言，手持机就是一个键盘了。

该值为 2，表示以虚拟串口方式连接到到电脑，手持机和电脑通信方式为串口，此时可以使用虚拟串口的收发函数，一般不使用，而使用 VirtualComOpen 函数。

该值为 3，表示以大容量磁盘方式连接到电脑，手持机表现为一个移动硬盘，可以访问 TF 卡内容。

UsbDisconnectToPC

`void UsbDisconnectToPC(void);`

功能：断开与电脑的连接

返回：无。

参数：无

注意：当不使用usb功能是需要断开连接以减少功耗

UsbMain

`void UsbMain(void);`

功能：调用系统通讯功能，与电脑进行文件传送，设置参数等功能

返回：无。

参数：无

注意：按ESC键可以退出该函数

VirtualComOpen

`void VirtualComOpen(void);`

功能：将USB虚拟成串口，

返回：无。

参数：无

注意：PC端需要安装相应的驱动程序

VirtualComClose

`void VirtualComClose(void);`

功能：关闭虚拟串口，

返回：无。

参数：无

注意：

VirtualComWrite

```
U32 VirtualComWrite(U8 *data, U32 size, U32 timeout);
```

功能： 往电脑端发送数据

参数： data需要发送的数据

size要发送的数据字节大小

timeout发送过程的的最大超时时间

返回： 实际发送的数据字节数

VirtualComRead

```
U32 VirtualComRead(U8 *data, U32 size, U32 timeout);
```

功能： 从电脑端读取数据

参数： data存储读取到的数据缓冲

size要读取的数据字节大小

timeout读取过程的的最大超时时间

返回： 实际读取的数据字节数

VirtualComRxString

```
U32 VirtualComRxString(U8 * data, U32 size, U32 Ftimeout, U32 Btimeout);
```

功能： 往电脑端发送数据

参数： data需要发送的数据

Size要发送的数据字节大小

Ftimeout发送过程的的最大超时时间

Btimeout字节间的间隔时间

返回： 实际发送的数据字节数

VirtualComTxString

```
U32 VirtualComTxString(U8 * data, U32 size, U32 Ftimeout, U32 Btimeout);
```

功能： 从电脑端读取数据

参数： data存储读取到的数据缓冲

size要读取的数据字节大小

Ftimeout读取过程的的最大超时时间

Btimeout字节间的最大超时时间

返回： 实际读取的数据字节数

SetUsbFuntion

```
void SetUsbFuntion(pUsbUserCallback Funtion, S32 type);
```

功能： 设置USB通讯时调用用户二次开发中的函数
参数： Funtion 要设置的功能函数
Type 函数对应的类型（上传或下载）

返回： 无

例子：

USER.H定义：

```
#define USB_USER_CALLBACK_UP 0  
#define USB_USER_CALLBACK_DOWN 1
```

上传：对应与动态库中 brGetMachInfo 函数

```
S32 UsbUserUpCallback(U32 index, U8 *Info) {  
    /*  
    //向电脑发送数据  
    memcpy(Info, "发送数据", 8);  
    */  
    return 8;  
}
```

/*

功能： 上传某个功能的参数，参数内容由用户决定
参数： index 某个功能，值从200~299
Info 要上传的内容（长度不超过32个字节）

返回： 内容长度，不超过32个字节，0表示内容为空

*/

```
SetUsbFuntion(UsbUserUpCallback, USB_USER_CALLBACK_UP);
```

下传：对应与动态库中 brSetMachInfo 函数

```
S32 UsbUserDownCallback(U32 index, U8 *Info) {  
    /*  
    //对收到的数据进行处理  
    LcdClear();  
    Lib_puts((S8*) Info);  
    */  
    return 0;  
}
```

/*

功能： 下传某个功能的参数，参数内容由用户决定
参数： index 某个功能，值从200~299
Info 要下传的内容（长度不超过32个字节）
长度由用户协议决定，比如下次的是字符串，那么长度就是info的字符串长度。
或者用户定义第一字节为长度，总之，整个内容长度不超过32字节。

返回： 函数操作成功或者失败，0表示成功，1表示失败

*/

```
SetUsbFuntion(UsbUserDownCallback, USB_USER_CALLBACK_DOWN);
```

字符编码转换

EncGetUtf8Size

```
int EncGetUtf8Size(unsigned char c);
```

功能:

根据字符的 UTF-8 编码的第一个字节求出该字符用 UTF-8 编码存储时需要多少个字节空间. 特殊地, 对于只占一个字节字符 (ASCII), 返回值为 0 .

参数:

c 字符的 UTF-8 编码的第一个字节的值

返回值:

该字符用 UTF-8 编码存储时需要多少个字节空间.

特殊地, 对于只占一个字节字符 (ASCII), 返回值为 0 .

EncUtf82UnicodeOne

```
int EncUtf82UnicodeOne(const unsigned char* pInput, unsigned long *Unic);
```

功能:

将一个字符的 UTF8 编码转换成 Unicode (UCS-2 和 UCS-4) 编码.

参数:

pInput 指向输入缓冲区, 以 UTF-8 编码

Unic 指向输出缓冲区, 其保存的数据即是 Unicode 编码值,
类型为 unsigned long .

返回值:

成功则返回该字符的 UTF8 编码所占用的字节数; 失败则返回 0.

注意:

1. UTF8 没有字节序问题, 但是 Unicode 有字节序要求;
字节序分为大端 (Big Endian) 和小端 (Little Endian) 两种;
在 Intel 处理器中采用小端法表示, 在此采用小端法表示. (低地址存低位)

EncUtf82Unicode

```
int EncUtf82Unicode(const unsigned char* pInput, int nMembIn, unsigned long* pOutput, int* nMembOut);
```

功能:

将字符串的 UTF8 编码转换成 Unicode (UCS-2 和 UCS-4) 编码.

参数:

pInput 指向输入缓冲区, 以 UTF-8 编码

nMembIn pInput 大小 (即, 字符串的长度) (单位: 1 字节)

pOutput 指向输出缓冲区, 用于保存 Unicode 编码值

nMembOut 输入: pOutput 可以存储的 Unicode 编码 (字符) 的个数 (单位: 4 字节);

输出：成功转换的字符的个数.

返回值：

若有错误发生，则返回 0 ；

若所有的字符都转换成功，则返回 1 ；

若输出缓冲区空间不足，导致只有部分字符转换成功，则返回 2 ；

另：*nMembOut 返回成功转换的字符的个数.

注意：

1. UTF8 没有字节序问题，但是 Unicode 有字节序要求；
字节序分为大端(Big Endian)和小端(Little Endian)两种；
在 Intel 处理器中采用小端法表示，在此采用小端法表示。（低地址存低位）

EncUtf82UnicodeStr

```
int EncUtf82UnicodeStr(const unsigned char *pInput, unsigned long *pOutput, int *nMembOut);
```

功能：

将以 0 结束的字符串的 UTF8 编码转换成 Unicode (UCS-2 和 UCS-4) 编码.

参数：

pInput 指向输入缓冲区，以 UTF-8 编码，以 0 结束.

pOutput 指向输出缓冲区，用于保存 Unicode 编码值

nMembOut 输入：pOutput 可以存储的 Unicode 编码(字符)的个数(单位：4 字节)；
输出：成功转换的字符的个数.

返回值：

若有错误发生，则返回 0 ；

若所有的字符都转换成功，则返回 1 ；

若输出缓冲区空间不足，导致只有部分字符转换成功，则返回 2 ；

另：*nMembOut 返回成功转换的字符的个数.

注意：

1. UTF8 没有字节序问题，但是 Unicode 有字节序要求；
字节序分为大端(Big Endian)和小端(Little Endian)两种；
在 Intel 处理器中采用小端法表示，在此采用小端法表示。（低地址存低位）

EncUnicode2Utf8One

```
int EncUnicode2Utf8One(unsigned long unic, unsigned char *pOutput, int outSize) ;
```

功能：

将一个字符的 Unicode (UCS-2 和 UCS-4) 编码转换成 UTF-8 编码.

参数：

unic 字符的 Unicode 编码值

pOutput 指向输出的用于存储 UTF8 编码值的缓冲区的指针

outsize pOutput 缓冲的大小

返回值：

返回转换后的字符的 UTF8 编码所占的字节数，如果出错则返回 0 .

注意:

1. UTF8 没有字节序问题, 但是 Unicode 有字节序要求;
字节序分为大端(Big Endian)和小端(Little Endian)两种;
在 Intel 处理器中采用小端法表示, 在此采用小端法表示. (低地址存低位)
2. 请保证 pOutput 缓冲区有最少有 6 字节的空间大小!

EncUnicode2Utf8

```
int EncUnicode2Utf8(const unsigned long *pInput, int nMemIn, unsigned char *pOutput, int *nMemOut) ;
```

功能:

将字符串的 Unicode(UCS-2 和 UCS-4)编码转换成 UTF8 编码.

参数:

pInput	指向输入缓冲区, 以 Unicode 编码
nMemIn	pInput 大小(即, 字符串的长度)(单位: 4 字节)
pOutput	指向输出缓冲区, 用于保存 UTF8 编码值
nMemOut	输入: pOutput 缓冲区的大小(单位: 1 字节); 输出: 成功转换后, UTF8 编码所占空间大小(单位: 1 字节).

返回值:

若有错误发生, 则返回 0 ;
若所有的字符都转换成功, 则返回 1 ;
若输出缓冲区空间不足, 导致只有部分字符转换成功, 则返回 2 ;
另: *nMemOut 返回 UTF8 编码所占空间大小(单位: 1 字节).

注意:

1. UTF8 没有字节序问题, 但是 Unicode 有字节序要求;
字节序分为大端(Big Endian)和小端(Little Endian)两种;
在 Intel 处理器中采用小端法表示, 在此采用小端法表示. (低地址存低位)

EncUnicode2Utf8Str

```
int EncUnicode2Utf8Str(const unsigned long *pInput, unsigned char *pOutput, int *nMemOut) ;
```

功能:

将字符串的 Unicode(UCS-2 和 UCS-4)编码(以 0 结束)转换成 UTF8 编码.

参数:

pInput	指向输入缓冲区, 以 Unicode 编码, 以 0 结束.
pOutput	指向输出缓冲区, 用于保存 UTF8 编码值
nMemOut	输入: pOutput 缓冲区的大小(单位: 1 字节); 输出: 成功转换后, UTF8 编码所占空间大小(单位: 1 字节).

返回值:

若有错误发生, 则返回 0 ;
若所有的字符都转换成功, 则返回 1 ;
若输出缓冲区空间不足, 导致只有部分字符转换成功, 则返回 2 ;
另: *nMemOut 返回 UTF8 编码所占空间大小(单位: 1 字节).

注意:

1. UTF8 没有字节序问题, 但是 Unicode 有字节序要求;
字节序分为大端(Big Endian)和小端(Little Endian)两种;
在 Intel 处理器中采用小端法表示, 在此采用小端法表示. (低地址存低位)

EncUtf82GbkStr

```
int EncUtf82GbkStr(const unsigned char *pInput, unsigned char *pOutput, int *nMembOut);
```

功能:

将以 0 结束的字符串的 UTF8 编码转换成 GBK 编码.

参数:

pInput	指向输入缓冲区, 以 UTF-8 编码, 以 0 结束.
pOutput	指向输出缓冲区, 用于保存 Unicode 编码值
nMembOut	输入: pOutput 可以存储的 Unicode 编码(字符)的个数(单位: 4 字节); 输出: 成功转换的字符的个数.

返回值:

若有错误发生, 则返回 0 ;
若所有的字符都转换成功, 则返回 1 ;
若输出缓冲区空间不足, 导致只有部分字符转换成功, 则返回 2 ;
另: *nMembOut 返回成功转换的字符的个数.

注意:

1. UTF8 没有字节序问题, 但是 Unicode 有字节序要求;
字节序分为大端(Big Endian)和小端(Little Endian)两种;
在 Intel 处理器中采用小端法表示, 在此采用小端法表示. (低地址存低位)

EncUnicode2GbkStr

```
int EncUnicode2GbkStr(const unsigned char *pInput, unsigned char *pOutput, int *nMembOut);
```

功能:

将以 0 结束的字符串的 Unicode 编码转换成 GBK 编码.

参数:

pInput	指向输入缓冲区, 以 UTF-8 编码, 以 0 结束.
pOutput	指向输出缓冲区, 用于保存 Unicode 编码值
nMembOut	输入: pOutput 可以存储的 Unicode 编码(字符)的个数(单位: 4 字节); 输出: 成功转换的字符的个数.

返回值:

若有错误发生, 则返回 0 ;
若所有的字符都转换成功, 则返回 1 ;
若输出缓冲区空间不足, 导致只有部分字符转换成功, 则返回 2 ;
另: *nMembOut 返回成功转换的字符的个数.

注意:

1. UTF8 没有字节序问题, 但是 Unicode 有字节序要求;

字节序分为大端 (Big Endian) 和小端 (Little Endian) 两种;
在 Intel 处理器中采用小端法表示, 在此采用小端法表示. (低地址存低位)

EncUnicode2GbkOne

```
unsigned short EncUnicode2GbkOne( unsigned short src );
```

功能:

将以字符 Unicode 编码转换成 GBK 编码.

参数:

Src 要转换的 Unicode 字符.

返回值:

GBK 字符

EncGbk2UnicodeOne

```
unsigned short EncGbk2UnicodeOne( unsigned short src );
```

功能:

将以字符 GBK 编码转换成 Unicode 编码.

参数:

Src 要转换的 GBK 字符.

返回值:

Unicode 字符

Ascii2Hex

```
int Ascii2Hex(S8 *O_data, U8 *N_data, int len)
```

功能: ASCII 到 HEX 的转换函数

参数: O_data: 转换数据的入口指针,

 N_data: 转换后新数据的入口指针

 len : 需要转换的长度

返回: -1: 转换失败, 其它: 转换后数据长度

注意: O_data[] 数组中的数据在转换过程中会被修改。

Hex2Ascii

```
int Hex2Ascii(U8 *O_data, S8 *N_data, int len)
```

功能: HEX 到 ASCII 的转换函数

参数: O_data: 转换数据的入口指针

 N_data: 转换后数据入口指针

 len : 需要转换的长度

返回： 转换后数据长度

LowerCase

```
void LowerCase(char *string);
```

功能： 将字符串转化为小写字符串

参数： string 需要转换的字符串

返回： 无

UpperCase

```
void UpperCase(char *string);
```

功能： 将字符串转化为大写字符串

参数： string 需要转换的字符串

返回： 无

TrimString

```
void TrimString (char *string);
```

功能： 去除字符串前面和后面的空格（0x20），回车（0x0D），换行（0x0A），Tab（0x09）字符

参数： string 需要处理的字符串，同时存储新的字符串。

返回： 无

```
unsigned int INPUT_BhChinese(int x0,int y0,signed short hParent);
```

```
unsigned int INPUT_PyChinese(int x0,int y0,signed short hParent);
```

```
char INPUT_English(int x0,int y0,signed short hParent);
```

```
GUI_BITMAP *GUI_GetIconAddress(int index,int type);
```

```
const GUI_FONT *GUI_GetFontAddress(int type);
```

```
const GUI_CURSOR *GUI_GetCursorAddress(int type,int size);
```

```
const GUI_BITMAP_METHODS * GUI_GetBitmapMethod(int type);
```

```
void LISTVIEW_Input(signed short hObj, int Column,int Row);
```

```
void EDIT_Input(signed short hObj);
```

```
void MULTIEDIT_Input(signed short hObj);
```

缓冲区 SQL 函数

SqlOpen

```
S32 SqlOpen(S8 *sql);
```

功能： 检查缓冲区是否接收正常，

参数： sql缓冲区地址

返回： 返回值小于0表示失败

SqlGotoRecord

S8 *SqlGotoRecord(U32 record, S8 *sql);

功能： 返回该记录的首地址

参数： record 记录号 (0~N)
sql缓冲区首地址

返回： 返回NULL 表示失败，其他表示该记录的首地址

SqlRecordRead

S32 SqlRecordRead(U32 record, S8 *data, S8 *sql);

功能： 读取某条记录

参数： record记录号 (0~N) 范围小于SqlRecordCount，
data用于存放该记录的缓冲区
sql缓冲区首地址

返回： 返回值小于0表示失败

SqlRecordCount

S32 SqlRecordCount(S8 *sql);

功能： 返回记录的总数

参数： sql缓冲区首地址

返回： 返回值小于0表示失败

SqlFieldCount

S32 SqlFieldCount(S8 *sql);

功能： 返回字段的总数

参数： sql缓冲区的首地址

返回： 返回值小于0表示失败

SqlFieldInfo

S32 SqlFieldInfo(U32 field, S8 *name, S8*sql);

功能： 获取字段信息

参数： field字段号 (0~M)，小于SqlFieldCount
name存放获取到的字段名称
sql为首地址，获取到的内容为字段名称

返回： 返回值小于0表示失败

SqlFieldGet

S32 SqlFieldGet(U32 record, U32 field, S8 *data, S8 *sql);

功能： 获取某个记录的某个字段内容

参数： record记录号 (0~N)
field字段号 (0~M)

data存放获取到的字段内容
返回： 返回值小于0表示失败

TF/SD 卡读写程序接口

函数名	描述
FatMount	注册/注销一个工作区
FatOpen	打开/创建一个文件
FatClose	关闭一个文件
FatRead	读取文件
FatWrite	写文件
FatLseek	移动读/写指针，扩展文件大小
FatTruncate	截断文件大小
FatSync	清空缓冲数据
FatOpenDir	打开一个目录
FatReadDir	读取一个目录项
FatGetfree	获取空闲簇
FatStat	获取文件状态
FatMkdir	创建一个目录
FatUnlink	删除一个文件或目录
FatChmod	修改属性
FatUtime	修改时间戳
FatRename	删除/移动一个文件或目录
FatChdir	修改当前目录
FatChdrive	修改当前驱动器
FatGetcwd	恢复当前目录
FatForward	直接输出文件数据流
FatMkfs	在驱动器上创建一个文件系统
FatFdisk	划分一个物理驱动器
FatGets	读取一个字符串
FatPutc	写一个字符
FatPuts	写一个字符串
FatPrintf	写一个格式化的字符串
FatTell	获取当前读/写指针
FatEof	测试一个文件是否到达文件末尾
FatSize	获取一个文件的大小
FatError	测试一个文件是否出错

FatMount

在FatFs 模块上注册/注销一个工作区(文件系统对象)。

```
FRESULT FatMount (  
    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

FatMount 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FatMount 函数后或存储介质改变后的第一次文件访问时完成的。

FatOpen

创建/打开一个用于访问文件的文件对象

```
FRESULT FatOpen (
    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**FatClose** 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**FatMount** 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。示
例(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 */
    FatMount(0, &fs[0]);
    FatMount(1, &fs[1]);

    /* 打开驱动器 1 上的源文件 */
    res = FatOpen(&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 = FatOpen(&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 = FatRead(&fsrc, buffer, sizeof(buffer), &br);
        if (res || br == 0) break;    /* 文件结束错误
    */ res = FatWrite(&fdst, buffer, br, &bw);
        if (res || bw < br) break;    /* 磁盘满错误 */
    }

    /* 关闭打开的文件
    */ FatClose(&fsrc);
    FatClose(&fdst);

    /* 注销工作区(在废弃前)
    */ FatMount(0, NULL);
    FatMount(1, NULL);
}
```


FatClose

关闭一个打开的文件

```
FRESULT FatClose (  
    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

FatClose 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。

FatRead

从一个文件读取数据

```
FRESULT FatRead (
    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。

FatWrite

写入数据到一个文件

```
FRESULT FatWrite (  
    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。

FatLseek

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 FatLseek (  
    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

FatLseek 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。**FatLseek** 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。

示例

```
/* 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 = FatLseek(file, 5000);  
  
    /* 移动文件读/写指针到文件结束处，以便添加数据 */  
res = FatLseek(file, file->fsize);  
    /* 向前3000 字节 */  
res = FatLseek(file, file->fptr + 3000);  
    /* 向后(倒带)2000 字节(注意溢出) */  
res = FatLseek(file, file->fptr - 2000);  
  
    /* 簇预分配(为了防止在流写时缓冲区上溢) */  
res = FatOpen(file, recfile, FA_CREATE_NEW | FA_WRITE); /* 创建一个文件  
    */ res = FatLseek(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 = FatLseek(file, DATA_START);          /* 没有簇分配延迟地记录数据流 */
```

```
...
    res = FatTruncate(file);          /* 截断未使用的区域 */
    res = FatLseek(file, 0);          /* 移动到文件起始处 */
...
res = FatClose(file);
```

FatTruncate

截断文件大小

```
FRESULT FatTruncate (  
    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

FatTruncate 函数当 **_FS_READONLY == 0** 并且 **_FS_MINIMIZE == 0** 时可用。

FatTruncate 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。

FatSync

冲洗一个写文件的缓存信息

```
FRESULT FatSync (
    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

FatSync 函数当 **_FS_READONLY == 0** 时可用。

FatSync 函数和 **FatClose** 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 **FatWrite** 后立即执行 **FatSync** 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 **FatClose** 前立即执行 **FatSync** 没有作用，因为在 **FatClose** 中执行了 **FatSync**。换句话说，这两个函数的差异就是文件对象是不是无效的。

FatOpenDir

打开一个目录

```
FRESULT FatOpenDir (  
    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

FatOpenDir 函数当_FS_MINIMIZE <= 1 时可用。

FatOpenDir 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。

FatReadDir

读取目录项

```
FRESULT FatReadDir (
    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

FatReadDir 函数当_FS_MINIMIZE <= 1 时可用。

FatReadDir 函数顺序读取目录项。目录中的所有项可以通过重复调用 FatReadDir 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FatReadDir 函数之前，文件信息结构中的lfname 和lfsz 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lfsz 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。

示例

```
FRESULT scan_files (
    char* path          /* Start node to be scanned (also used as work area) */
)
{
    FRESULT res;
    FILINFO fno;

    DIR dir;
    int i;

    char *fn; /* This function is assuming non-Unicode cfg. */
```

```
#if _USE_LFN
```

```
    static char lfn[_MAX_LFN + 1];
```

```
    fno.lfname = lfn;
```

```
    fno.lfsize = sizeof lfn;
```

```
#endif
```

```
    res = FatOpenDir(&dir, path);
```

```
/* Open the directory */
```

```
    if (res == FR_OK) {
```

```
        i = strlen(path);
```

```
        for (;;) {
```

```
            res = FatReadDir(&dir, &fno);
```

```
/* Read a directory item */
```

```
            if (res != FR_OK || fno.fname[0] == 0) break; /* Break on error or end of dir */
```

```
            if (fno.fname[0] == '.') continue; /* Ignore dot entry */
```

```
#if _USE_LFN
```

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

```
#else
```

```
        fn = fno.fname;
```

```
#endif
```

```
        if (fno.fattrib & AM_DIR) {
```

```
/* It is a directory */
```

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

```
            res = scan_files(path);
```

```
            if (res != FR_OK) break;
```

```
            path[i] = 0;
```

```
        } else {
```

```
/* It is a file. */
```

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

```
        }
```

```
    }
```

```
}
```

```
    return res;
```

```
}
```

FatGetfree

获取空闲簇的数目

```
FRESULT FatGetfree (  
    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 以 '\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

FatGetfree 函数当 _FS_READONLY == 0 并且 _FS_MINIMIZE == 0 时有效。

FatGetfree 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。

示例

```
FATFS *fs;  
    DWORD fre_clust, fre_sect, tot_sect;  
  
    /* Get drive information and free clusters */ res  
    = FatGetfree("/", &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);
```

FatStat

获取文件状态

```
FRESULT FatStat (  
    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

FatStat 函数当_FS_MINIMIZE == 0 时可用。

FatStat 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FatReadDir 函数。

FatMkdir

创建一个目录

```
FRESULT FatMkdir (  
    const XCHAR* DirName /* 目录名的指针 */  
);
```

参数 DirName 以 '\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

FatMkdir 函数当 _FS_READONLY == 0 并且 _FS_MINIMIZE == 0 时可用。

FatMkdir 函数创建一个新目录。

示例

```
res = FatMkdir("sub1");  
if (res) die(res);  
  
res = FatMkdir("sub1/sub2");  
if (res) die(res);  
  
res = FatMkdir("sub1/sub2/sub3");  
if (res) die(res);
```

FatUnlink

移除一个对象

```
FRESULT FatUnlink (  
    const XCHAR* FileName /* 对象名的指针 */  
);
```

参数 FileName 以 '\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

FatUnlink 函数当 _FS_READONLY == 0 并且 _FS_MINIMIZE == 0 时可用。

FatUnlink 函数移除一个对象。不要移除打开的对象或当前目录。

FatChmod

修改一个文件或目录的属性。

```
FRESULT FatChmod (  
    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 以 '\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

FatChmod 函数当 `_FS_READONLY == 0` 并且 `_FS_MINIMIZE == 0` 时可用。

FatChmod 函数修改一个文件或目录的属性。

示例

```
// 设置只读标志，清除存档标志，其他不变  
FatChmod("file.txt", AR_RDO, AR_RDO | AR_ARC);
```

FatUtime

FatUtime 函数修改一个文件或目录的时间戳。

```
FRESULT FatUtime (  
    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 以'\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

FatUtime 函数当_FS_READONLY == 0 并且_FS_MINIMIZE == 0 时可用。

FatUtime 函数修改一个文件或目录的时间戳。

FatRename

重命名一个对象。

```
FRESULT FatRename (  
    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

FatRename 函数当_FS_READONLY == 0 并且_FS_MINIMIZE == 0 时可用。

FatRename 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。 示例

```
/* 重命名一个对象 */  
FatRename("oldname.txt", "newname.txt");  
  
/* 重命名并且移动一个对象到另一个目录 */  
FatRename("oldname.txt", "dir1/newname.txt");
```

FatChdir

FatChdir 函数改变一个驱动器的当前目录。

```
FRESULT FatChdir (  
    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

FatChdir 函数当_FS_RPATH == 1 时可用。

FatChdir 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。

示例

```
// 改变当前驱动器的当前目录(根目录下的 dir1)  
FatChdir("/dir1");  
  
// 改变驱动器2 的当前目录(父目录)  
FatChdir("2:..");
```

FatChdrive

FatChdrive 函数改变当前驱动器。

```
FRESULT FatChdrive (  
    BYTE Drive /* 逻辑驱动器号 */  
);
```

参数

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

FatChdrive 函数当_FS_RPATH == 1 时可用。

FatChdrive 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。

FatGetcwd

恢复当前目录。

```
FRESULT FatGetcwd (  
    TCHAR* Buffer, /* Pointer to the buffer */  
    UINT BufferLen /* The length of the buffer */  
);
```

参数 Buffer——指向接收当前目录字符串的缓冲区
BufferLen——缓冲区的大小，单位为 TCHAR
返回值

FR_OK (0) 函数成功。

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。

FR_TIMEOUT 函数由于线程安全控制超时而退出。(相关选项: _TIMEOUT)

FR_NOT_ENOUGH_CORE 没有足够的内存进行操作。原因有:

- 不能为 LFN 工作缓冲区分配内存 (相关选项: _USE_LFN)
- 得到的表大小不能满足要求的大小。

描述

FatGetcwd 函数用完整的路径字符串 (包括驱动器号) 来恢复当前驱动器上的当前目录。

提示

在 _FS_RPATH == 2 时可用。

FatForward

读取文件数据并将其转发到数据流设备。

```
FRESULT FatForward (
    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

FatForward 函数当_USE_FORWARD == 1 并且_FS_TINY == 1 时可用。

FatForward 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。 示例(音频播放)

```
/*-----*/
/* 示例代码：数据传输函数，将被FatForward 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
FatForward 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FatForward 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 = FatOpen(&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 = FatForward(&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;
}

```

FatMkfs

在驱动器上创建一个文件系统

```
FRESULT FatMkfs (  
    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

FatMkfs 函数当_FS_READOLNY == 0 并且_USE_MKFS == 1 时可用。

FatMkfs 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。

FatFdisk

划分一个物理驱动器。

```
FRESULT FatFdisk (  
    BYTE Drive, /* Physical drive number */  
    const DWORD Partitions[], /* Partition size */  
    void* Work /* Work area */  
);
```

参数

Drive——指定要划分的物理驱动器 **Partitions[]**——分区映象表,必须有四个项目。 **Work**——指向函数工作区的指针。其大小必须至少为 **_MAX_SS** 字节。

返回值

FR_OK (0)函数成功。

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_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_INVALID_PARAMETER 所给参数无效或不一致。

描述

FatFdisk 函数创建一个分区表到物理驱动器的 **MBR**。分区规则为通用 **FDISK** 格式,所以可以创建多达四个主分区。不支持扩展分区。**Partitions[]**指定了如何划分物理驱动器。第一个项目指定第一个主分区的大小,第四个项目指定第四个主分区。如果其值小于或等于100,表示分区占整个磁盘空间的百分比。如果大于100,则表示以扇区为单位的分区大小。

提示

在 **_FS_READOLNY == 0**、**_USE_MKFS == 1** 并且 **_MULTI_PARTITION == 2** 时可用。

示例

```
/* Volume management table defined by user (required when _MULTI_PARTITION != 0) */ PARTITION  
VolToPart[] = {  
    {0, 1}, /* Logical drive 0 ==> Physical drive 0, 1st partition */  
    {0, 2}, /* Logical drive 1 ==> Physical drive 0, 2nd partition */  
    {1, 0} /* Logical drive 2 ==> Physical drive 1, auto detection */  
};  
/* Initialize a brand-new disk drive mapped to physical drive 0 */ FATFS  
Fatfs;  
DWORD plist[] = {50, 50, 0, 0}; /* Divide drive into two partitions */  
BYTE work[_MAX_SS];  
FatFdisk(0, plist, work); /* Divide physical drive 0 */  
FatMount(0, &Fatfs);  
FatMkfs(0, 0, 0); /* Create an FAT volume on the logical drive 0. 2nd argument is ignored. */  
FatMount(0, 0);
```



```
FatMount(1,      &Fatfs);  
FatMkfs(1, 0, 0);  
FatMount(1, 0);
```

FatGets

FatGets 从文件中读取一个字符串。

```
char* FatGets (  
    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

FatGets 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。

FatGets 函数是FatRead 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，FatGets()返回一个空字符串。
可以使用宏FatEof()和FatError()检查EOF 和错误状态。

FatPutc

FatPutc 函数向文件中写入一个字符。

```
int FatPutc (  
    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

FatPutc 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。

FatPutc 函数是FatWrite 的一个封装函数。

FatPuts

FatPuts 函数向文件中写入一个字符串。

```
int FatPuts (  
    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

FatPuts()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。

FatPuts()是FatPutc()的一个封装函数。

FatPrintf

FatPrintf 函数向文件中写入一个格式化字符串。

```
int FatPrintf (
    FIL* FileObject,    /* 文件对象指针 */
    const char* Format, /* 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

FatPrintf 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。

FatPrintf 函数是FatPutc 和FatPuts 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：1
- 标志：0

示例

```
FatPrintf(&fil, "%d", 1234);           /* "1234" */
FatPrintf(&fil, "%6d,%3d%%", -200, 5); /* "-200, 5%" */
FatPrintf(&fil, "%-6u", 100);           /* "100   " */
FatPrintf(&fil, "%ld", 12345678L);       /* "12345678" */
FatPrintf(&fil, "%04x", 0xA3);           /* "00a3" */
FatPrintf(&fil, "%08LX", 0x123ABC);       /* "00123ABC" */
FatPrintf(&fil, "%016b", 0x550F);        /* "0101010100001111" */
FatPrintf(&fil, "%s", "String");        /* "String" */

FatPrintf(&fil, "%-4s", "abc");          /* "abc  " */
FatPrintf(&fil, "%4s", "abc");           /* " abc" */
FatPrintf(&fil, "%c", 'a');              /* "a" */
FatPrintf(&fil, "%f", 10.0);             /* FatPrintf lacks floating point support */
```

FatTell

获取一个文件的当前读/写指针。

```
DWORD FatTell (  
    FIL* FileObject    /* File object */  
);
```

参数 FileObject——指向打开文件对象结构的指针。 返回值

返回文件的当前读/写指针。 描述

在这个版本里，FatTell 函数是以一个宏来实现的。

```
#define FatTell(fp) ((fp)->fptr)
```

提示 总是可用。

FatEof

测试一个文件的文件末尾。

```
int FatEof (  
    FILE* FileObject    /* File object */  
);
```

参数 FileObject——指向打开文件对象结构的指针。 返回值

如果读/写指针到达文件末尾，FatEof 函数返回一个非零值；否则返回0。

描述

在这个版本里，FatEof 函数是以一个宏来实现的。

```
#define FatEof(fp) (((fp)->fptr) == ((fp)->fsize) ? 1 : 0)
```

提示 总是
可用。

FatSize

获取一个文件的大小。

```
DWORD FatSize (  
    FIL* FileObject    /* File object */  
);
```

参数 FileObject——指向打开文件对象结构的指针。 返回值

返回文件的大小，单位为字节。

描述

在这个版本里，FatSize 函数是以一个宏来实现的。

```
#define FatSize(fp) ((fp)->fsize)
```

提示 总是
可用。

FatError

测试文件是否出错。

```
int FatError (  
    FIL* FileObject    /* File object */  
);
```

参数 FileObject——指向打开文件对象结构的指针。 返回值

如果有错误返回非零值；否则返回 0。

描述

在这个版本里，FatError 函数是以一个宏来实现的。

```
#define FatError(fp) (((fp)->flag & FA_ERROR) ? 1 : 0)
```

提示 总是
可用。

五、外设功能

1、Barcode 条码(一维、二维)

条码扫描功能的使用很简单，只要使用 `BarcodeGetID` 函数打开条码扫描引擎，然后就判断返回结果就可以。

```
U32 BarcodeGetID(S8 *Buffer,int Maxlen,int Timeout);
```

功能：驱动扫描引擎读取条码信息

返回：0 表示读取失败，非0表示读取成功，此时表示读取到数据长度，如果返回值大于等于Maxlen，说明Buffer缓冲区不够大，数据不够存放。

参数：

1) **Buffer** 存放条码的数据的缓冲区

2) **Maxlen** 缓冲区的大小，一般情况下 buffer 的大小需要根据条码的长度而定，一维码可以取 32 字节，二维码可以取 128 字节。

2) **Timeout** 读取超时的时间，不宜过长，一般设置为 3000（3 秒）

注意：

因为扫描引擎本身会有功耗管理，一般读取几秒扫描不到有效信息，会自动关闭红光输出，这个是为了延长寿命和降低功耗。系统也会对扫描引擎的供电做管理，如果连续5分钟用户没有触发扫描功能，将自动关闭引擎供电，以节省功耗。调用`BarcodeGetID` 会自动重新工作。

2、Bluetooth 蓝牙

蓝牙功能的使用很简单，一般是作为串口使用，只要使用 `UartOpen`、函数打开蓝牙功能，`UartClose` 关闭蓝牙，利用 `UartWrite`、`UartRead` 等几个函数就可以简单进行扫描读取数据。

例如：

```
sUART *pBluetooth;
```

```
打开蓝牙：pBluetooth=UartOpen(9600,UART_MODE_8B_NONE_1S,DEVICE_BLUETOOTH);
```

注意：

由于蓝牙是点对点通信，使用前需要进行配对，配对成功后，重新打开蓝牙功能，此时就可以当作普通的串口使用，默认的工作波特率是9600，该波特率只和手持机有关，和连接的设备无关。我们提供配对程序，可将模块配对为主模式或者从模式。

3、Camera 摄像

```
U32 PhotoPreview (int x0,int y0);  
U32 PhotoShoot (S8 * Filename);
```

4、GPRS 通信功能

```
int GprsInit(void);
```

功能：初始化GPRS模块任务

返回：0 表示成功。

参数：无

注意：

改函数初始了gprs功能的后台任务，该任务负责GPRS模块的启动，初始化和异常处理的工作。

```
int GprsTcpConnect (S8* nAddress, S8*nPort);
```

功能：连接到指定的TCP服务器地址和端口信息

返回：0 表示成功，负数表示失败及原因

参数：

1) **nAddress** 地址或者域名，如 www.cldata.net

2) **nPort** 端口号，如 2000

注意：

该函数必须在模块初始化结束后才可以调用，否则返回正在初始化的错误。

```
int GprsTcpTx(U8 *data, int len);
```

功能：发送数据到服务器

返回：实际发送的数据长度，负数表示失败及原因

参数：

1) **data** 需要发送的数据

2) **len** 需要发送的数据长度

注意：

该函数必须在模块连接上服务器时才可以调用，否则返回发送错误。

```
int GprsTcpRx(U8 * data, int len, int timeout);
```

功能：从服务器接受数据

返回：实际接受的的数据长度

参数：

1) **data** 接收到的数据数据存放的缓冲区

2) **len** 需要接收的数据长度

3) **timeout** 接收超时时间

注意:

该函数必须在模块连接上服务器时才可以调用，否则返回发送错误。

```
int GprsHttpGet(char *url, char *reply, int maxlen);
```

功能: 使用HTTP协议中GET方式发送数据到服务器

返回: 服务器响应的数据长度，负数表示失败

参数:

1) **url** 数据连接，如 `http://www.cldata.net/gettest.asp?k=1`

2) **reply** 存放服务器发回的数据缓冲区

3) **maxlen** 缓冲区大小

注意:

该函数必须在模块初始化结束后才可以调用，否则返回正在初始化的错误。

```
int GprsHttpPost(char *url, U8*data, int datalen, char *reply, int maxlen);
```

功能: 使用HTTP协议中POST方式发送数据到服务器

返回: 服务器响应的数据长度，负数表示失败

参数:

1) **url** 数据连接，如 `http://www.cldata.net/gettest.asp?k=1`

2) **datalen** 需要发送的数据长度

3) **reply** 存放服务器发回的数据缓冲区

4) **maxlen** 缓冲区大小

注意:

该函数必须在模块初始化结束后才可以调用，否则返回正在初始化的错误。

```
int GprsNetSignal(void);
```

功能: 获取当前的网络信号强度及模块状态

返回: 信号强度

参数:

无

注意:

该函数必须在模块初始化结束后才可以调用，否则返回正在初始化的错误。

```
int GprsSMSSend(char *phoneno, char *text);
```

功能: 发送短信

返回: 0表示成功，负数表示失败

参数:

1) **phoneno** 手机号码

2) **text** 需要发送的数据

注意:

该函数必须在模块初始化结束后才可以调用，否则返回正在初始化的错误。

5、GPS 全球定位系统

U32 GPSGetPosition (S8 *Buffer, int Maxlen, int Timeout);

功能：获取经纬度

返回：0 表示读取失败，非0表示读取成功，此时表示读取到数据长度，如果返回值大于等于Maxlen，说明Buffer缓冲区不够大，数据不够存放。

参数：

- 1) Buffer 存放经纬度数据的缓冲区，经纬度通过逗号分隔，如 23.11, 112.33
- 2) Maxlen 缓冲区的大小，一般可以用 32 字节
- 2) Timeout 读取超时的时间，不宜过长，一般设置为 3000（3 秒）

注意：

系统进行有效定位后会自动关闭电源以节省能源，当调用该函数后会进行再次定位，如果之前以及有效定位，会比较快的返回位置值，否则会超时返回。用户多次调用该函数可使GPS定位持续工作。

6、IRDA_H 高速红外

高速红外功能的使用很简单，只要使用 UartOpen、函数打开功能，UartClose 关闭，利用 UartWrite、UartRead 等几个函数就可以简单进行数据收发。

例如：

sUART * pIrdah;

打开：pIrdah=UartOpen(9600,UART_MODE_8B_NONE_1S, DEVICE_HIGHT_IRDA);

注意：

高速红外的通信距离一般为米左右，主要用于连接微型打印机，或者调试具有同样功能的红外设备

7、IRDA_L 载波红外 38.4K

低速载波红外功能的使用很简单，只要使用 UartOpen、函数打开功能，UartClose 关闭，利用 UartWrite、UartRead 等几个函数就可以简单进行数据收发。

例如：

sUART * pIrdah;

打开：pIrdah=UartOpen(9600,UART_MODE_8B_NONE_1S, DEVICE_LOW_IRDA);

注意：

低速红外的通信距离一般为十米左右，主要用于电表，水表，和其他具有该功能的设备进行通信，读取数据和设置参数。

8、RFID 低频电子标签

RFID11784GetID

U32 RFID11784GetID (S8 *Buffer, int Maxlen, int Timeout);

功能：驱动低频RFID模块工作

返回：0 表示读取失败，非0表示读取成功，此时表示读取到数据长度，如果返回值大于等于Maxlen，说明Buffer缓冲区不够大，数据不够存放。

参数：

1) Buffer 存放条码的数据的缓冲区

2) Maxlen 缓冲区的大小，一般情况下 buffer 的大小需要根据条码的长度而定，一维码可以取 32 字节，二维码可以取 128 字节。

2) Timeout 读取超时的时间，不宜过长，一般设置为 3000（3 秒）

注意：

系统也会对扫描引擎的供电做管理，如果连续5分钟用户没有触发读取功能，将自动关闭引擎供电，以节省功耗。调用Rfid125KGetID会自动重新工作。

9、RFID 高频频电子标签

对于简单的应用，我们提供下面函数，该函数获取 ID 卡的唯一卡号，如果需要进行数据块的读写，请参考相关文档。

RFID14443AGetID

U32 RFID14443AGetID (S8 *Buffer, int Maxlen, int Timeout);

功能：驱动低频RFID模块工作

返回：0 表示读取失败，非0表示读取成功，此时表示读取到数据长度，如果返回值大于等于Maxlen，说明Buffer缓冲区不够大，数据不够存放。

参数：

1) Buffer 存放条码的数据的缓冲区

2) Maxlen 缓冲区的大小，一般情况下 buffer 的大小需要根据条码的长度而定，一维码可以取 32 字节，二维码可以取 128 字节。

2) Timeout 读取超时的时间，不宜过长，一般设置为 3000（3 秒）

注意:

系统也会对扫描引擎的供电做管理，如果连续5分钟用户没有触发读取功能，将自动关闭引擎供电，以节省功耗。调用RFID14443AGetID会自动重新工作。

RFID14443BGetID

U32 IS014443BGetID (S8 *Buffer,int Maxlen,int Timeout);

功能: 驱动低频RFID模块工作

返回: 0 表示读取失败，非0表示读取成功，此时表示读取到数据长度，如果返回值大于等于Maxlen，说明Buffer缓冲区不够大，数据不够存放。

参数:

- 1) Buffer 存放条码的数据的缓冲区
- 2) Maxlen 缓冲区的大小，一般情况下 buffer 的大小需要根据条码的长度而定，一维码可以取 32 字节，二维码可以取 128 字节。
- 2) Timeout 读取超时的时间，不宜过长，一般设置为 3000（3 秒）

注意:

系统也会对扫描引擎的供电做管理，如果连续5分钟用户没有触发读取功能，将自动关闭引擎供电，以节省功耗。调用RFID14443BGetID会自动重新工作。

RFID15693GetID

U32 RFID15693GetID (S8 *Buffer,int Maxlen,int Timeout);

功能: 驱动低频RFID模块工作

返回: 0 表示读取失败，非0表示读取成功，此时表示读取到数据长度，如果返回值大于等于Maxlen，说明Buffer缓冲区不够大，数据不够存放。

参数:

- 1) Buffer 存放条码的数据的缓冲区
- 2) Maxlen 缓冲区的大小，一般情况下 buffer 的大小需要根据条码的长度而定，一维码可以取 32 字节，二维码可以取 128 字节。
- 2) Timeout 读取超时的时间，不宜过长，一般设置为 3000（3 秒）

注意:

系统也会对扫描引擎的供电做管理，如果连续5分钟用户没有触发读取功能，将自动关闭引擎供电，以节省功耗。调用RFID15693GetID会自动重新工作。

ISO14443A 数据读写函数

`void ISO14443AOpen(void)`

功能：驱动ISO14443A模块工作

返回：无

参数：无

`void ISO14443AClose(void)`

功能：关闭ISO14443A模块电源

返回：无

参数：无

`char ISO14443ARequest(unsigned char req_code,unsigned char *pTagType);`

功 能：寻卡

参数说明: req_code[IN]:寻卡方式

0x52 = 寻感应区内所有符合 14443A 标准的卡

0x26 = 寻未进入休眠状态的卡

pTagType[OUT]: 卡片类型代码

0x4400 = Mifare_UltraLight

0x0400 = Mifare_One(S50)

0x0200 = Mifare_One(S70)

0x0800 = Mifare_Pro(X)

0x4403 = Mifare_DESFire

返 回: 成功返回 MI_OK

`char ISO14443AAnticoll(unsigned char *pSnr);`

功 能：防冲撞

参数说明: pSnr[OUT]:卡片序列号，4 字节

返 回: 成功返回 MI_OK

`char ISO14443ASelect(unsigned char *pSnr);`

功 能：选定卡片

参数说明: pSnr[IN]:卡片序列号，4 字节

返 回: 成功返回 MI_OK

`char ISO14443AAuthState(unsigned char auth_mode,unsigned char addr,unsigned char *pKey,unsigned char *pSnr);`

功 能：验证卡片密码

参数说明: auth_mode[IN]: 密码验证模式

0x60 = 验证 A 密钥

0x61 = 验证 B 密钥

addr[IN]: 块地址

pKey[IN]: 密码
pSnr[IN]: 卡片序列号, 4 字节
返 回: 成功返回 MI_OK

char ISO14443ARead(unsigned char addr,unsigned char *pData);

功 能: 读取 M1 卡一块数据
参数说明: addr[IN]: 块地址
pData[OUT]: 读出的数据, 16 字节
返 回: 成功返回 MI_OK

char ISO14443AWrite(unsigned char addr,unsigned char *pData);

功 能: 写数据到 M1 卡一块
参数说明: addr[IN]: 块地址
pData[IN]: 写入的数据, 16 字节
返 回: 成功返回 MI_OK

char ISO14443AHalt(void);

功 能: 命令卡片进入休眠状态
返 回: 成功返回 MI_OKAPI 返回值

void ISO14443AAntennaOn(void)

功 能: 开启天线
返 回: 无
注意: 每次启动或关闭天线发射之间应至少有 1ms 的间隔

void ISO14443AAntennaOff(void)

功 能: 关闭天线
返 回: 无

char ISO14443AValue(unsigned char dd_mode,unsigned char addr,unsigned char *pValue)

功 能: 扣款和充值
参数说明: dd_mode[IN]: 命令字
0xC0 = 扣款
0xC1 = 充值
addr[IN]: 钱包地址
pValue[IN]: 4 字节增(减)值, 低位在前
返 回: 成功返回 MI_OK

char ISO14443ABakValue(unsigned char sourceaddr, unsigned char goaladdr)

功 能: 备份钱包
参数说明: sourceaddr[IN]: 源地址
goaladdr[IN]: 目标地址
返 回: 成功返回 MI_OK

错误宏定义

宏定义	错误码	含义
MI_OK	0	成功
MI_NOTAGERR	1	没有卡
MI_ERR	2	失败

测试例程:

```
unsigned char Wdata[16] =
{0x12, 0x34, 0x56, 0x78, 0xED, 0xCB, 0xA9, 0x87, 0x12, 0x34, 0x56, 0x78, 0x01, 0xFE, 0x01, 0xFE};
unsigned char Rdata[16] = {0,};
```

//M1 卡的某一块写为如下格式，则该块为钱包，可接收扣款和充值命令

//4 字节金额（低字节在前）+4 字节金额取反+4 字节金额+1 字节块地址+1 字节块地址
取反+1 字节块地址+1 字节块地址取反

```
unsigned char ValueData[4] = {0x12,0,0,0};
unsigned char DefaultKey[6] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
unsigned char CardType[2];
unsigned char CardID[4];
```

```
int RfidTestMain(void)
{
    unsigned char status;

    ISO14443AOpen ();

    while ( 1 )
    {
        memset(CardType,0,sizeof(CardType));
        memset(CardID,0,sizeof(CardID));
        ISO14443AAntennaOn();
        status = ISO14443ARequest(PICC_REQALL, CardType);
        if (status != MI_OK)
        {
            LedMiddle(0);
            continue;
        }
        LedMiddle(1);           //检测到有卡存在

        status = ISO14443AAnticoll(CardID);
        if (status != MI_OK)
        {
            continue;
        }
        Bell(100,0);return MI_OK; //读取到卡号
        status = ISO14443ASelect(CardID);
        if (status != MI_OK)
        {
            continue;
        }
    }
}
```

```

        status = ISO14443AAuthState(PICC_AUTHENT1A, 1, DefaultKey, CardID);
        if (status != MI_OK)
        {
            continue;
        }

        status = ISO14443AWrite(1, Wdata);
        if (status != MI_OK)
        {
            continue;
        }

        status = ISO14443AValue(PICC_DECREMENT, 1, ValueData);
        if (status != MI_OK)
        {
            continue;
        }

        status = ISO14443ABakValue(1, 2);
        if (status != MI_OK)
        {
            continue;
        }

        status = ISO14443ARead(2, Rdata);
        if (status != MI_OK)
        {
            continue;
        }
        Bell(100,0); return MI_OK;

        status = ISO14443AHalt();
        if (status != MI_OK)
        {
            continue;
        }

        Bell(100,0);

        return MI_OK;
    }
}

```

1、初始化端口

函数原型：

```
U32 ISOOpenCom ();
```

功能：

打开通信端口。

输入参数： 无

输出参数：

端口打开成功返回“0”，否则为非“0”值。

2、关闭端口

函数原型:

```
U32 ISOCloseCom ();
```

功能:

关闭通信端口。

输入参数:

无

输出参数:

端口关闭成功返回“0”，否则为非“0”值。

3、指定设备标识

函数原型:

```
U32 SetDeviceNumber(U16 DiveID);
```

功能:

设置读写器的设备标识。

输入参数:

DiveID - 设备标识，如“0x0101”

输出参数:

设置成功返回“0”，否则为非“0”值。

4、获取设备标识

函数原型:

```
U32 GetDeviceNumber(U16 *pDevID);
```

功能:

读取读写器的设备标识。

输入参数:

pDevID - 设备标识，如得到标识号“0x0101”时，格式化成字符串为“0101”;

输出参数:

读取成功返回“0”，否则为非“0”值。

5、读取硬件版本号

函数原型:

```
U32 GetHardwareVersion (U16 , U16 *pVersion);
```

功能:

读取读写器的硬件版本号。

输入参数:

DiveID - 设备标识,

输出参数:

pVersion - 硬件版本号, 如得到版本号“0x0101”时, 格式化字符串为“0101”;

读取成功返回 “0”, 否则为非 “0” 值。

6、读写器工作模式

函数原型:

```
U32 SetWorkMode(U16 DiveID, U8 bType);
```

功能:

设置读写器非接触工作模式 ISO14443-A 或者 ISO14443-B。

输入参数:

DiveID - 设备标识

bType - 0x00: 设置为 TYPE_A 方式
 0x01: 设置为 TYPE_B 方式
 0x02: 设置为 AT88RF020 卡方式
 0x03: 设置为 ISO15693 卡方式

输出参数:

设置成功返回 “0”, 否则为非 “0” 值。

7、设置天线状态

函数原型:

```
U32 SetAntennaStatus(U16 DiveID, U8 bMode);
```

功能:

设置读写器天线的工作模式。

输入参数:

DiveID - 设备标识
bMode - 0x00:关闭天线
 0x01:开启天线

输出参数:

设置成功返回“0”，否则为非“0”值。

8、寻 Mifare 卡

函数原型:

U32 ISO14443_3ARequest(U16 DiveID, U8 bMode, U8 *pTagType, U8 *pLength);

功能:

寻在读写范围内的 Mifare 卡。

输入参数:

DiveID - 设备标识
bMode - 0x26: 寻未进入休眠状态的卡
 0x52: 寻所有状态的卡

输出参数:

pTagType - 返回寻到的卡类型
pLength - 返回数据的长度;
设置成功返回“0”，否则为非“0”值。

9、Mifare 卡防冲突

函数原型:

U32 ISO14443_3AAnticoll(U16 DiveID, U8 *pSnr, U8* pLength);

功能:

防冲突。

输入参数:

DiveID - 设备标识

输出参数:

pSnr - 返回卡序列号

pLength - 返回数据的长度；
设置成功返回“0”，否则为非“0”值。

10、 Mifare 选卡

函数原型:

U32 ISO14443_3ASelect(U16 DiveID, U8 *pSnr, U8 bLen, U8 *pLength);

功能:

选取某张 Mifare 卡。

输入参数:

DiveID - 设备标识
pSnr - 卡序列号
bLen - 卡序列号长度

输出参数:

pLength - 返回 1 字节卡容量；
选择成功返回“0”，否则为非“0”值。

11、 休眠 Mifare 卡

函数原型:

U32 ISO14443_3AHalt(U16 DiveID);

功能:

使被激活的某张 Mifare 卡处于休眠状态。

输入参数:

DiveID - 设备标识

输出参数:

设置成功返回“0”，否则为非“0”值。

12、 Mifare 卡认证

函数原型:

U32 ISO14443_3AAuthentication2(U16 DiveID, U8 bMode, U8 bBlock, U8 *pKey);

功能:

对 Mifare 卡的某一块进行认证操作。

输入参数:

- DiveID - 设备标识
- bMode - 密钥属性, 0x60 = 'A', 0x61 = 'B'
- bBlock - 绝对块号
- pKey - 6 字节密钥

输出参数:

认证成功返回“0”，否则为非“0”值。

13、 Mifare 卡读卡

函数原型:

```
U32 ISO14443_3ARead(U16 DiveID, U8 bBlock, U8 *pData);
```

功能:

读取 Mifare 卡的某一块。

输入参数:

- DiveID - 设备标识
- bBlock - 绝对块号

输出参数:

- pData - 返回 16 字节数据
- 读取成功返回“0”，否则为非“0”值。

14、 Mifare 写卡

函数原型:

```
U32 ISO14443_3AWrite(U16 DiveID, U8 bBlock, U8 *pData);
```

功能:

对 Mifare 某一块进行写数据操作。

输入参数:

- DiveID - 设备标识
- bBlock - 绝对块号
- pData - 16 字节数据

输出参数:

写入成功返回“0”，否则为非“0”值。

15、 Mifare 钱包初始化(暂无)

函数原型:

```
U32 ISO14443_3APurseInit(U16 DiveID, U8 bBlock, long lValue);
```

功能:

将 Mifare 卡某一块初始化成钱包结构。

输入参数:

DiveID	-	设备标识
bBlock	-	绝对块号
lValue	-	初始钱包值

输出参数:

初始化成功返回“0”，否则为非“0”值。

16、 读取 Mifare 钱包值(暂无)

函数原型:

```
U32 ISO14443_3AReadVal(U16 DiveID, U8 bBlock, long *plValue);
```

功能:

读取 Mifare 钱包。

输入参数:

DiveID	-	设备标识
bBlock	-	绝对块号

输出参数:

plValue	-	返回钱包值
---------	---	-------

正确成功返回“0”，否则为非“0”值。

17、 Mifare 钱包扣款(暂无)

函数原型:

```
U32 ISO14443_3ADecrement(U16 DiveID, U8 bBlock, long lValue);
```

功能:

对 Mifare 钱包进行扣款操作。

输入参数:

DiveID	-	设备标识
bBlock	-	绝对块号
IValue	-	待扣钱包值

输出参数:

成功扣款返回“0”，否则为非“0”值。

18、 Mifare 钱包增值(暂无)

函数原型:

U32 ISO14443_3AIncrement(U16 DiveID, U8 bBlock, long IValue);

功能:

对 Mifare 钱包进行增值操作。

输入参数:

DiveID	-	设备标识
bBlock	-	绝对块号
IValue	-	待增钱包值

输出参数:

成功增款返回“0”，否则为非“0”值。

19、 Mifare 数据回传(暂无)

函数原型:

U32 ISO14443_3ARestore(U16 DiveID, U8 bBlock);

功能:

对 Mifare 将指定的块的内容传入一段缓冲区中。

输入参数:

DiveID	-	设备标识
bBlock	-	绝对块号

输出参数:

成功回传返回“0”，否则为非“0”值。

20、 Mifare 数据传送(暂无)

函数原型:

```
U32 ISO14443_3ATransfer(U16 DiveID, U8 bBlock);
```

功能:

对 Mifare 将缓冲区中的数据内容到指定块中。

输入参数:

DiveID	-	设备标识
bBlock	-	绝对块号

输出参数:

成功传送返回“0”，否则为非“0”值。

21、 Utri Light 选卡

函数原型:

```
U32 ULSelect(U16 DiveID, U8 *pSnr);
```

功能:

选取某张 Utri Light 卡。

输入参数:

DiveID	-	设备标识
--------	---	------

输出参数:

pSnr	-	返回 7 字节卡序列号
------	---	-------------

选择成功返回“0”，否则为非“0”值。

22、 Utri Light 写卡

函数原型:

```
U32 ULWrite(U16 DiveID, U8 bPage, U8 *pData, U8 *pRev);
```

功能:

将 16 个字节的数据写入到以某页开始的四块中。

输入参数:

DiveID	-	设备标识
bPage	-	将要写入的开始页

pData - 将要写入的 16 字节数据

输出参数:

pRev - 返回 7 字节卡序列号

写入成功返回“0”，否则为非“0”值。

23、 ISO15693_Inventory

函数原型:

U32 ISO15693_Inventory(U16 DiveID, U8 *pData, U8 *pLength);

功能:

。

输入参数:

DiveID - 设备标识

输出参数:

pData - 返回 9 字节的数据，1 字节的 DSFID 和 8 字节的 UID

pLength - 返回数据的长度

读取成功返回“0”，否则为非“0”值。

24、 ISO15693_Stay_Quiet

函数原型:

U32 ISO15693_Stay_Quiet(U16 DiveID, U8 *pUID);

功能:

输入参数:

DiveID - 设备标识

pUID - 8 字节的 UID

输出参数:

成功返回“0”，否则为非“0”值。

25、 选择 15693

函数原型:

U32 ISO15693_Select(U16 DiveID, U8 *pUID);

功能:

选择一张 15693 卡

输入参数:

DiveID - 设备标识

pUID - 8 字节的 UID

输出参数:

选择成功返回“0”，否则为非“0”值。

26、 ISO15693_ResetToReady

函数原型:

U32 ISO15693_ResetToReady(U16 DiveID, U8 bMode, U8 *pUID);

功能:

输入参数:

DiveID - 设备标识

bMode - bit0=Select_flags,bit1=Addres_flags

pUID - 8 字节的 UID

输出参数:

成功返回“0”，否则为非“0”值。

27、 读取 15693 中的数据

函数原型:

U32 ISO15693Read(U16 DiveID, U8 bMode, U8 *pUID, U8 bBlock, U8 *pData, U8 *pLength);

功能:

读取 15693 指定块后的指定行数的数据

输入参数:

DiveID - 设备标识

bMode - bit0=Select_flags,bit1=Addres_flags

pUID - 8 字节的 UID

bBlock - 块号

输出参数:

- pData - 读取后返回的数据
 - pLength - 返回读取数据的长度
- 读取成功返回“0”，否则为非“0”值。

28、 写入数据到 15693 卡中

函数原型:

```
U32 ISO15693Write(U16 DiveID, U8 bMode, U8 *pUID, U8 bBlock, U8 *pData);
```

功能:

向 15693 卡写入 4 字节的数据

输入参数:

- DiveID - 设备标识
- bMode - bit0=Select_flags, bit1=Address_flags
- pUID - 8 字节的 UID
- bBlock - 块号
- pData - 待写入的 4 字节的数据

输出参数:

写入成功返回“0”，否则为非“0”值。

29、 锁定 15693 卡某块

函数原型:

```
U32 ISO15693LockBlock(U16 DiveID, U8 bMode, U8 *pUID, U8 bBlock);
```

功能:

锁定 15693 卡中的某块数据

输入参数:

- DiveID - 设备标识
- bMode - bit0=Select_flags, bit1=Address_flags
- pUID - 8 字节的 UID
- bBlock - 块号

输出参数:

锁定成功返回“0”，否则为非“0”值。

30、 写入 15693 的 AFI

函数原型:

```
U32 ISO15693WriteAFI(U16 DiveID, U8 bMode, U8 *pUID, U8 bAFI);
```

功能:

向 15693 卡中写入 AFI

输入参数:

DiveID	-	设备标识
bMode	-	bit0=Select_flags,bit1=Addres_flags
pUID	-	8 字节的 UID
bAFI	-	AFI 数据

输出参数:

写入成功返回“0”，否则为非“0”值。

31、 锁定 15693 的 AFI

函数原型:

```
U32 ISO15693LockAFI(U16 DiveID, U8 bMode, U8 *pUID);
```

功能:

锁定 15693 卡中的 AFI

输入参数:

DiveID	-	设备标识
bMode	-	bit0=Select_flags,bit1=Addres_flags
pUID	-	8 字节的 UID

输出参数:

锁定成功返回“0”，否则为非“0”值。

32、 写入 15693 的 DSFID

函数原型:

```
U32 ISO15693WriteDSFID(U16 DiveID, U8 bMode, U8 *pUID, U8 bDSFID);
```

功能:

写入 15693 卡中的 DSFID

输入参数:

- DiveID - 设备标识
- bMode - bit0=Select_flags,bit1=Addres_flags
- pUID - 8 字节的 UID
- bDSFID - 1 字节的 DSFID

输出参数:

写入成功返回“0”，否则为非“0”值。

33、 写入 15693 的 DSFID

函数原型:

```
U32 ISO15693LockDSFID(U16 DiveID, U8 bMode, U8 *pUID);
```

功能:

锁定 15693 卡中的 DSFID

输入参数:

- DiveID - 设备标识
- bMode - bit0=Select_flags,bit1=Addres_flags
- pUID - 8 字节的 UID

输出参数:

锁定成功返回“0”，否则为非“0”值。

34、 读取 15693 的卡片信息

函数原型:

```
U32 ISO15693GetSystemInformation(U16 DiveID, U8 bMode,U8 *pUID, U8 *pDSFID, U8 *pAFI);
```

功能:

读取 15693 卡中的卡片信息

输入参数:

- DiveID - 设备标识
- bMode - bit0=Select_flags,bit1=Addres_flags

输出参数:

- pUID - 8 字节的 UID
- pDSFID - 1 字节

pAFI - 1 字节

读取成功返回“0”，否则为非“0”值。

返回的卡片数据的 UID 第七个字节决定后续的操作方式变量“bMode”。

35、 读取 15693 的某块数据信息

函数原型:

```
U32 ISO15693GetBlockSecurity(U16 DiveID, U8 bMode, U8 *pUID, U8 bBlock, U8 *pData, U8 *pLength);
```

功能:

读取 15693 卡中从指定的某块开始的指定的块的数据信息

输入参数:

- DiveID - 设备标识
- bMode - bit0=Select_flags,bit1=Address_flags
- pUID - 8 字节的 UID
- bBlock - 指定的开始块

输出参数:

- pData - 读取的数据信息
 - pLength - 数据信息的长度
- 读取成功返回“0”，否则为非“0”值。

1 API 返回值

宏定义	错误码	含义
LIB_SUCCESS	0	操作成功
ERROR_WRITE_COM	1	写数据失败
ERROR_WRITE_COM_FAILED	2	写数据出错
ERROR_READ_COM	3z	读数据失败
ERROR_CHECK	4	返回数据检验失败
ERROR_PARAM	5	传入接口参数错误

10、RFID 超高频频电子标签

U32 RFID180006GetID (S8 *Buffer,int Maxlen,int Timeout);

功能：驱动低频RFID模块工作

返回：0 表示读取失败，非0表示读取成功，此时表示读取到数据长度，如果返回值大于等于Maxlen，说明Buffer缓冲区不够大，数据不够存放。

参数：

1) Buffer 存放条码的数据的缓冲区

2) Maxlen 缓冲区的大小，一般情况下 buffer 的大小需要根据条码的长度而定，一维码可以取 32 字节，二维码可以取 128 字节。

2) Timeout 读取超时的时间，不宜过长，一般设置为 3000（3 秒）

注意：

系统也会对扫描引擎的供电做管理，如果连续5分钟用户没有触发读取功能，将自动关闭引擎供电，以节省功耗。调用RFID180006GetID会自动重新工作。

11、RS232 串行数据接口

RS232 功能的使用很简单，只要使用 UartOpen、函数打开功能，UartClose 关闭，利用 UartWrite、UartRead 等几个函数就可以简单进行扫描读取数据。

例如：

sUART *pRs232;

打开：pRs232=UartOpen(9600,UART_MODE_8B_NONE_1S, DEVICE_RS232);

注意：

RS232的电平是正负12V，与电脑的串口电平一致，不能够和TTL电平或者CMOS电平的设备连接，如果连接的设备电平不匹配，会造成烧毁芯片。

12、RS485 串行数据接口

RS232 功能的使用很简单，只要使用 UartOpen、函数打开功能，UartClose 关闭，利用 UartWrite、UartRead 等几个函数就可以简单进行扫描读取数据。

例如：

sUART *pRs485;

打开：pRs485=UartOpen(9600,UART_MODE_8B_NONE_1S, DEVICE_RS485);

注意：

RS485为半双工工作方式，不能够同时收发。

13、Voice 中文语音播读

S32 TextToVoice(char *Text, int Speed, int Volume);

功能：将文本转换为语音输出

返回：

参数：

- | | |
|-----------|-----------|
| 1) Text | 需要播读的文本文件 |
| 2) Speed | 播放的语速 |
| 2) Volume | 播放的音量 |

注意：

系统也会对扫描引擎的供电做管理，如果连续5分钟用户没有触发播放功能，将自动关闭引擎供电，以节省功耗。调用TextToVoice会自动重新工作。

14、WIFI(Wireless Fidelity)

```
int WifiHttpGet(char *url, char *result, int *maxlen);
int WifiFtpSetParam(char *serverip, char *port, char *user, char *pass);
int WifiFtpSetDir(char *dir);
int WifiFtpGet(char *srcfilename, char *disfilename);
int WifiFtpPut(char *filename);
int WifiTcpTxData(U8 *data, int len);
int WifiTcpRxData(U8 * data, U32 size, U32 Ftimeout, U32 Btimeout);
```

六、硬件函数

1、中断服务

BSP_IntVectSet

```
void BSP_IntVectSet (CPU_DATA int_id, CPU_FNCT_VOID isr);
```

可以替换中断服务程序。例如：BSP_IntVectSet(BSP_INT_ID_USART3, WifiRxIrQServer);

2、秒定时调用

设置秒中断功能

SetSecondIRQ

```
void SetSecondIRQ(pExFunction Funtion);
```

参数说明:

Funtion 要在中断内部运行的函数, 该函数原型为:

```
void funtion (void);
```

该函数没有输入参数和返回参数, 每一秒钟执行一次. 可以用于显示时间和电量等.

例如:

```
#include "GUI.h"
```

```
#include "User.h"
```

```
void test(void){
```

```
    S8 timebuf[20];
```

```
    U8  year,month,day,hour, minute,second;
```

```
        RtcGetDate(&year,&month, &day);
```

```
        RtcGetTime(&hour, &minute, &second);
```

```
        Lib_sprintf(timebuf,"%2d月%2d号 %2d:%2d:%2d",month,day,hour,minute,second);
```

```
        if(timebuf[4]==' ')timebuf[4]='0';
```

```
        if(timebuf[12]==' ')timebuf[12]='0';
```

```
        if(timebuf[15]==' ')timebuf[15]='0';
```

```
        LcdMoveto(10,80);Lib_printf(timebuf);
```

```
    }
```

```
int main(void){
```

```
    SetSecondIRQ(test);
```

```
    while(1){
```

```
        if(KeyWait()==KEY_ESC) return 0;
```

```
    }
```

```
}
```

3、UART

4、SPI

5、IIC

6、ADC

7、DAC

8、TIM

9、SDIO

10、JTAG

11、IO

硬件功能的使用，可以参考 STM32 硬件库的相关文档

七、OS 多任务系统

1、参考 OS 中文说明书.pdf

八、GUI 图形界面

1、参考 GUI 中文说明书.pdf

九、附录

GB2312 符号表

【A1A1】	【A1A2】、	【A1A3】。	【A1A4】•
【A1A5】 -	【A1A6】 ∼	【A1A7】 ∴	【A1A8】 ″
【A1A9】々	【A1AA】—	【A1AB】～	【A1AC】∥
【A1AD】…	【A1AE】‘	【A1AF】’	【A1B0】“
【A1B1】”	【A1B2】{	【A1B3】}	【A1B4】〈
【A1B5】〉	【A1B6】《	【A1B7】》	【A1B8】[
【A1B9】]	【A1BA】『	【A1BB】』	【A1BC】⌈
【A1BD】⌋	【A1BE】【	【A1BF】】	【A1C0】±
【A1C1】×	【A1C2】÷	【A1C3】：	【A1C4】∧
【A1C5】√	【A1C6】Σ	【A1C7】Π	【A1C8】∪
【A1C9】∩	【A1CA】∈	【A1CB】∴	【A1CC】√
【A1CD】⊥	【A1CE】//	【A1CF】∠	【A1D0】∩
【A1D1】⊙	【A1D2】∫	【A1D3】\$	【A1D4】≡
【A1D5】≅	【A1D6】≈	【A1D7】≈	【A1D8】∞
【A1D9】≠	【A1DA】⋈	【A1DB】⋈	【A1DC】≤

【A1DD】 \geq	【A1DE】 ∞	【A1DF】 $\ddot{\cdot}$	【A1E0】 \therefore
【A1E1】 \dagger	【A1E2】 $\ddot{\cdot}$	【A1E3】 $^{\circ}$	【A1E4】 $'$
【A1E5】 $"$	【A1E6】 $^{\circ}\text{C}$	【A1E7】 $\text{\$}$	【A1E8】 $\text{\textcircled{X}}$
【A1E9】 $\text{\textcircled{C}}$	【A1EA】 \textsterling	【A1EB】 \textpermyriad	【A1EC】 \textsection
【A1ED】 N_{0}	【A1EE】 \star	【A1EF】 \blackstar	【A1F0】 \bigcirc
【A1F1】 \bullet	【A1F2】 \odot	【A1F3】 \diamond	【A1F4】 \blacklozenge
【A1F5】 \square	【A1F6】 \blacksquare	【A1F7】 \triangle	【A1F8】 \blacktriangle
【A1F9】 \otimes	【A1FA】 \rightarrow	【A1FB】 \leftarrow	【A1FC】 \uparrow
【A1FD】 \downarrow	【A1FE】 $=$		

【A2A1】i	【A2A2】ii	【A2A3】iii	【A2A4】iv
【A2A5】v	【A2A6】vi	【A2A7】vii	【A2A8】viii
【A2A9】ix	【A2AA】x	【A2AB】	【A2AC】
【A2AD】	【A2AE】	【A2AF】	【A2B0】
【A2B1】1.	【A2B2】2.	【A2B3】3.	【A2B4】4.
【A2B5】5.	【A2B6】6.	【A2B7】7.	【A2B8】8.
【A2B9】9.	【A2BA】10.	【A2BB】11.	【A2BC】12.
【A2BD】13.	【A2BE】14.	【A2BF】15.	【A2C0】16.
【A2C1】17.	【A2C2】18.	【A2C3】19.	【A2C4】20.
【A2C5】(1)	【A2C6】(2)	【A2C7】(3)	【A2C8】(4)
【A2C9】(5)	【A2CA】(6)	【A2CB】(7)	【A2CC】(8)
【A2CD】(9)	【A2CE】(10)	【A2CF】(11)	【A2D0】(12)
【A2D1】(13)	【A2D2】(14)	【A2D3】(15)	【A2D4】(16)
【A2D5】(17)	【A2D6】(18)	【A2D7】(19)	【A2D8】(20)
【A2D9】①	【A2DA】②	【A2DB】③	【A2DC】④
【A2DD】⑤	【A2DE】⑥	【A2DF】⑦	【A2E0】⑧
【A2E1】⑨	【A2E2】⑩	【A2E3】 \texteuro	【A2E4】
【A2E5】 \rightarrow	【A2E6】 \rightleftarrows	【A2E7】 \rightleftarrows	【A2E8】(四)
【A2E9】(五)	【A2EA】(六)	【A2EB】(七)	【A2EC】(八)
【A2ED】(九)	【A2EE】(+)	【A2EF】	【A2F0】
【A2F1】I	【A2F2】II	【A2F3】III	【A2F4】IV
【A2F5】V	【A2F6】VI	【A2F7】VII	【A2F8】VIII
【A2F9】IX	【A2FA】X	【A2FB】XI	【A2FC】XII
【A2FD】	【A2FE】		

【A3A1】!	【A3A2】"	【A3A3】#	【A3A4】 \textyen
【A3A5】%	【A3A6】&	【A3A7】'	【A3A8】(
【A3A9】)	【A3AA】*	【A3AB】+	【A3AC】,
【A3AD】—	【A3AE】.	【A3AF】/	【A3B0】0
【A3B1】1	【A3B2】2	【A3B3】3	【A3B4】4
【A3B5】5	【A3B6】6	【A3B7】7	【A3B8】8
【A3B9】9	【A3BA】:	【A3BB】;	【A3BC】<
【A3BD】=	【A3BE】>	【A3BF】?	【A3C0】@
【A3C1】A	【A3C2】B	【A3C3】C	【A3C4】D
【A3C5】E	【A3C6】F	【A3C7】G	【A3C8】H
【A3C9】I	【A3CA】J	【A3CB】K	【A3CC】L
【A3CD】M	【A3CE】N	【A3CF】O	【A3D0】P

【A3D1】 Q	【A3D2】 R	【A3D3】 S	【A3D4】 T
【A3D5】 U	【A3D6】 V	【A3D7】 W	【A3D8】 X
【A3D9】 Y	【A3DA】 Z	【A3DB】 [【A3DC】 \
【A3DD】]	【A3DE】 ^	【A3DF】 _	【A3E0】 `
【A3E1】 a	【A3E2】 b	【A3E3】 c	【A3E4】 d
【A3E5】 e	【A3E6】 f	【A3E7】 g	【A3E8】 h
【A3E9】 i	【A3EA】 j	【A3EB】 k	【A3EC】 l
【A3ED】 m	【A3EE】 n	【A3EF】 o	【A3F0】 p
【A3F1】 q	【A3F2】 r	【A3F3】 s	【A3F4】 t
【A3F5】 u	【A3F6】 v	【A3F7】 w	【A3F8】 x
【A3F9】 y	【A3FA】 z	【A3FB】 {	【A3FC】
【A3FD】 }	【A3FE】 —		

【A4A1】 あ	【A4A2】 あ	【A4A3】 い	【A4A4】 い
【A4A5】 う	【A4A6】 う	【A4A7】 え	【A4A8】 え
【A4A9】 お	【A4AA】 お	【A4AB】 か	【A4AC】 が
【A4AD】 き	【A4AE】 ぎ	【A4AF】 く	【A4B0】 ぐ
【A4B1】 け	【A4B2】 げ	【A4B3】 こ	【A4B4】 ご
【A4B5】 さ	【A4B6】 ざ	【A4B7】 し	【A4B8】 じ
【A4B9】 す	【A4BA】 ず	【A4BB】 せ	【A4BC】 ぜ
【A4BD】 そ	【A4BE】 ぞ	【A4BF】 た	【A4C0】 だ
【A4C1】 ち	【A4C2】 ぢ	【A4C3】 っ	【A4C4】 っ
【A4C5】 づ	【A4C6】 て	【A4C7】 で	【A4C8】 と
【A4C9】 ど	【A4CA】 な	【A4CB】 に	【A4CC】 ぬ
【A4CD】 ね	【A4CE】 の	【A4CF】 は	【A4D0】 ば
【A4D1】 ぱ	【A4D2】 ひ	【A4D3】 び	【A4D4】 び
【A4D5】 ふ	【A4D6】 ぶ	【A4D7】 ぷ	【A4D8】 へ
【A4D9】 べ	【A4DA】 ぺ	【A4DB】 ほ	【A4DC】 ぼ
【A4DD】 ぽ	【A4DE】 ま	【A4DF】 み	【A4E0】 む
【A4E1】 め	【A4E2】 も	【A4E3】 や	【A4E4】 や
【A4E5】 ゆ	【A4E6】 ゆ	【A4E7】 よ	【A4E8】 よ
【A4E9】 ら	【A4EA】 り	【A4EB】 る	【A4EC】 れ
【A4ED】 ろ	【A4EE】 わ	【A4EF】 わ	【A4F0】 ゐ
【A4F1】 ゑ	【A4F2】 を	【A4F3】 ん	【A4F4】
【A4F5】	【A4F6】	【A4F7】	【A4F8】
【A4F9】	【A4FA】	【A4FB】	【A4FC】
【A4FD】	【A4FE】		

【A5A1】 ア	【A5A2】 ア	【A5A3】 イ	【A5A4】 イ
【A5A5】 ウ	【A5A6】 ウ	【A5A7】 エ	【A5A8】 エ
【A5A9】 オ	【A5AA】 オ	【A5AB】 カ	【A5AC】 ガ
【A5AD】 キ	【A5AE】 ギ	【A5AF】 ク	【A5B0】 グ
【A5B1】 ケ	【A5B2】 ゲ	【A5B3】 コ	【A5B4】 ゴ
【A5B5】 サ	【A5B6】 ザ	【A5B7】 シ	【A5B8】 ジ
【A5B9】 ス	【A5BA】 ズ	【A5BB】 セ	【A5BC】 ゼ
【A5BD】 ソ	【A5BE】 ゾ	【A5BF】 タ	【A5C0】 ダ
【A5C1】 チ	【A5C2】 デ	【A5C3】 ツ	【A5C4】 ツ

【A5C5】 ツ	【A5C6】 テ	【A5C7】 デ	【A5C8】 ト
【A5C9】 ド	【A5CA】 ナ	【A5CB】 ニ	【A5CC】 ヌ
【A5CD】 ネ	【A5CE】 ノ	【A5CF】 ハ	【A5D0】 バ
【A5D1】 パ	【A5D2】 ヒ	【A5D3】 ビ	【A5D4】 ピ
【A5D5】 フ	【A5D6】 ブ	【A5D7】 プ	【A5D8】 ヘ
【A5D9】 ベ	【A5DA】 ペ	【A5DB】 ホ	【A5DC】 ボ
【A5DD】 ポ	【A5DE】 マ	【A5DF】 ミ	【A5E0】 ム
【A5E1】 メ	【A5E2】 モ	【A5E3】 ヤ	【A5E4】 ヤ
【A5E5】 ユ	【A5E6】 ユ	【A5E7】 ヨ	【A5E8】 ヨ
【A5E9】 ラ	【A5EA】 リ	【A5EB】 ル	【A5EC】 レ
【A5ED】 ロ	【A5EE】 ヲ	【A5EF】 ワ	【A5F0】 ヰ
【A5F1】 エ	【A5F2】 チ	【A5F3】 ン	【A5F4】 ヱ
【A5F5】 カ	【A5F6】 ケ	【A5F7】	【A5F8】
【A5F9】	【A5FA】	【A5FB】	【A5FC】
【A5FD】	【A5FE】		

【A6A1】 Α	【A6A2】 Β	【A6A3】 Γ	【A6A4】 Δ
【A6A5】 Ε	【A6A6】 Ζ	【A6A7】 Η	【A6A8】 Θ
【A6A9】 Ι	【A6AA】 Κ	【A6AB】 Λ	【A6AC】 Μ
【A6AD】 Ν	【A6AE】 Ξ	【A6AF】 Ο	【A6B0】 Π
【A6B1】 Ρ	【A6B2】 Σ	【A6B3】 Τ	【A6B4】 Υ
【A6B5】 Φ	【A6B6】 Χ	【A6B7】 Ψ	【A6B8】 Ω
【A6B9】	【A6BA】	【A6BB】	【A6BC】
【A6BD】	【A6BE】	【A6BF】	【A6C0】
【A6C1】 α	【A6C2】 β	【A6C3】 γ	【A6C4】 δ
【A6C5】 ε	【A6C6】 ζ	【A6C7】 η	【A6C8】 θ
【A6C9】 ι	【A6CA】 κ	【A6CB】 λ	【A6CC】 μ
【A6CD】 ν	【A6CE】 ξ	【A6CF】 ο	【A6D0】 π
【A6D1】 ρ	【A6D2】 σ	【A6D3】 τ	【A6D4】 υ
【A6D5】 φ	【A6D6】 χ	【A6D7】 ψ	【A6D8】 ω
【A6D9】 ’	【A6DA】 °	【A6DB】 `	【A6DC】 ∴
【A6DD】 ∴	【A6DE】 !	【A6DF】 ?	【A6E0】 ∩
【A6E1】 ∩	【A6E2】 ∪	【A6E3】 ∪	【A6E4】 ∩
【A6E5】 ∩	【A6E6】 ≧	【A6E7】 ≧	【A6E8】 ∩
【A6E9】 ⊥	【A6EA】 ≡	【A6EB】 ≡	【A6EC】 ≡
【A6ED】 ⊆	【A6EE】 ⊆	【A6EF】 ⊆	【A6F0】 ∩
【A6F1】 ∩	【A6F2】	【A6F3】 ∴	【A6F4】
【A6F5】 {	【A6F6】	【A6F7】	【A6F8】
【A6F9】	【A6FA】	【A6FB】	【A6FC】
【A6FD】	【A6FE】		

【A7A1】 Α	【A7A2】 Β	【A7A3】 Β	【A7A4】 Γ
【A7A5】 Д	【A7A6】 Е	【A7A7】 Ё	【A7A8】 Ж
【A7A9】 З	【A7AA】 И	【A7AB】 Й	【A7AC】 К
【A7AD】 Л	【A7AE】 М	【A7AF】 Н	【A7B0】 О
【A7B1】 П	【A7B2】 Р	【A7B3】 С	【A7B4】 Т
【A7B5】 У	【A7B6】 Ф	【A7B7】 Х	【A7B8】 Ц

【A7B9】	Ч	【A7BA】	Ш	【A7BB】	Щ	【A7BC】	Ъ
【A7BD】	Ы	【A7BE】	Ь	【A7BF】	Э	【A7C0】	Ю
【A7C1】	Я	【A7C2】		【A7C3】		【A7C4】	
【A7C5】		【A7C6】		【A7C7】		【A7C8】	
【A7C9】		【A7CA】		【A7CB】		【A7CC】	
【A7CD】		【A7CE】		【A7CF】		【A7D0】	
【A7D1】	а	【A7D2】	б	【A7D3】	в	【A7D4】	г
【A7D5】	д	【A7D6】	е	【A7D7】	ё	【A7D8】	ж
【A7D9】	з	【A7DA】	и	【A7DB】	й	【A7DC】	к
【A7DD】	л	【A7DE】	м	【A7DF】	н	【A7E0】	о
【A7E1】	п	【A7E2】	р	【A7E3】	с	【A7E4】	т
【A7E5】	у	【A7E6】	ф	【A7E7】	х	【A7E8】	ц
【A7E9】	ч	【A7EA】	ш	【A7EB】	щ	【A7EC】	ъ
【A7ED】	ы	【A7EE】	ь	【A7EF】	э	【A7F0】	ю
【A7F1】	я	【A7F2】		【A7F3】		【A7F4】	
【A7F5】		【A7F6】		【A7F7】		【A7F8】	
【A7F9】		【A7FA】		【A7FB】		【A7FC】	
【A7FD】		【A7FE】					

【A8A1】	ā	【A8A2】	á	【A8A3】	ă	【A8A4】	à
【A8A5】	ē	【A8A6】	é	【A8A7】	ě	【A8A8】	è
【A8A9】	ī	【A8AA】	í	【A8AB】	ï	【A8AC】	ì
【A8AD】	ō	【A8AE】	ó	【A8AF】	ö	【A8B0】	ò
【A8B1】	ū	【A8B2】	ú	【A8B3】	ŭ	【A8B4】	ù
【A8B5】	ŭ	【A8B6】	ű	【A8B7】	ű	【A8B8】	ű
【A8B9】	ü	【A8BA】	ê	【A8BB】	α	【A8BC】	ḡ
【A8BD】	ń	【A8BE】	ň	【A8BF】	ñ	【A8C0】	g
【A8C1】		【A8C2】		【A8C3】		【A8C4】	
【A8C5】	ㄅ	【A8C6】	ㄆ	【A8C7】	ㄇ	【A8C8】	ㄏ
【A8C9】	ㄏ	【A8CA】	ㄊ	【A8CB】	ㄋ	【A8CC】	ㄌ
【A8CD】	ㄋ	【A8CE】	ㄍ	【A8CF】	ㄆ	【A8D0】	ㄏ
【A8D1】	ㄍ	【A8D2】	ㄇ	【A8D3】	ㄏ	【A8D4】	ㄏ
【A8D5】	ㄏ	【A8D6】	ㄏ	【A8D7】	ㄏ	【A8D8】	ㄏ
【A8D9】	ㄏ	【A8DA】	ㄏ	【A8DB】	ㄏ	【A8DC】	ㄏ
【A8DD】	ㄏ	【A8DE】	ㄏ	【A8DF】	ㄏ	【A8E0】	ㄏ
【A8E1】	ㄏ	【A8E2】	ㄏ	【A8E3】	ㄏ	【A8E4】	ㄏ
【A8E5】	ㄏ	【A8E6】	ㄏ	【A8E7】	ㄏ	【A8E8】	ㄏ
【A8E9】	ㄏ	【A8EA】		【A8EB】		【A8EC】	
【A8ED】		【A8EE】		【A8EF】		【A8F0】	
【A8F1】		【A8F2】		【A8F3】		【A8F4】	
【A8F5】		【A8F6】		【A8F7】		【A8F8】	
【A8F9】		【A8FA】		【A8FB】		【A8FC】	
【A8FD】		【A8FE】					

【A9A1】		【A9A2】		【A9A3】		【A9A4】	—
【A9A5】	—	【A9A6】		【A9A7】		【A9A8】	---
【A9A9】	---	【A9AA】	∴	【A9AB】	∴	【A9AC】	----

【A9AD】----	【A9AE】⋮	【A9AF】⋮	【A9B0】┐
【A9B1】┐	【A9B2】┐	【A9B3】┐	【A9B4】┐
【A9B5】┐	【A9B6】┐	【A9B7】┐	【A9B8】┐
【A9B9】┐	【A9BA】┐	【A9BB】┐	【A9BC】┐
【A9BD】┐	【A9BE】┐	【A9BF】┐	【A9C0】┐
【A9C1】┐	【A9C2】┐	【A9C3】┐	【A9C4】┐
【A9C5】┐	【A9C6】┐	【A9C7】┐	【A9C8】┐
【A9C9】┐	【A9CA】┐	【A9CB】┐	【A9CC】┐
【A9CD】┐	【A9CE】┐	【A9CF】┐	【A9D0】┐
【A9D1】┐	【A9D2】┐	【A9D3】┐	【A9D4】┐
【A9D5】┐	【A9D6】┐	【A9D7】┐	【A9D8】┐
【A9D9】┐	【A9DA】┐	【A9DB】┐	【A9DC】┐
【A9DD】┐	【A9DE】┐	【A9DF】┐	【A9E0】┐
【A9E1】┐	【A9E2】┐	【A9E3】┐	【A9E4】┐
【A9E5】┐	【A9E6】┐	【A9E7】┐	【A9E8】┐
【A9E9】┐	【A9EA】┐	【A9EB】┐	【A9EC】┐
【A9ED】┐	【A9EE】┐	【A9EF】┐	【A9F0】
【A9F1】	【A9F2】	【A9F3】	【A9F4】
【A9F5】	【A9F6】	【A9F7】	【A9F8】
【A9F9】	【A9FA】	【A9FB】	【A9FC】
【A9FD】	【A9FE】		

Keyboard/Keypad 键值表

Usage ID(Dec) 十进制值	Usage ID(Hex) 十六进制值	Usage Name 按键功能名称
0	00	Reserved (no event indicated)
1	01	Keyboard ErrorRollOver
2	02	Keyboard POSTFail
3	03	Keyboard ErrorUndefined
4	04	Keyboard a and A
5	05	Keyboard b and B
6	06	Keyboard c and C
7	07	Keyboard d and D4
8	08	Keyboard e and E
9	09	Keyboard f and F
10	0A	Keyboard g and G
11	0B	Keyboard h and H
12	0C	Keyboard i and I
13	0D	Keyboard j and J
14	0E	Keyboard k and K
15	0F	Keyboard l and L
16	10	Keyboard m and M
17	11	Keyboard n and N
18	12	Keyboard o and O

19	13	Keyboard p and P
20	14	Keyboard q and Q
21	15	Keyboard r and R
22	16	Keyboard s and S
23	17	Keyboard t and T
24	18	Keyboard u and U
25	19	Keyboard v and V
26	1A	Keyboard w and W
27	1B	Keyboard x and X
28	1C	Keyboard y and Y
29	1D	Keyboard z and Z
30	1E	Keyboard 1 and !
31	1F	Keyboard 2 and @
32	20	Keyboard 3 and #
33	21	Keyboard 4 and \$
34	22	Keyboard 5 and %
35	23	Keyboard 6 and ^
36	24	Keyboard 7 and &
37	25	Keyboard 8 and *
38	26	Keyboard 9 and (
39	27	Keyboard 0 and)
40	28	Keyboard Return (ENTER)
41	29	Keyboard ESCAPE
42	2A	Keyboard DELETE (Backspace)
43	2B	Keyboard Tab
44	2C	Keyboard Spacebar
45	2D	Keyboard - and (underscore)
46	2E	Keyboard = and +
47	2F	Keyboard [and {
48	30	Keyboard] and }
49	31	Keyboard \ and
50	32	Keyboard Non-US # and ~
51	33	Keyboard ; and :
52	34	Keyboard ' and "
53	35	Keyboard Grave Accent and Tilde
54	36	Keyboard, and <
55	37	Keyboard . and >
56	38	Keyboard / and ?
57	39	Keyboard Caps Lock
58	3A	Keyboard F1
59	3B	Keyboard F2
60	3C	Keyboard F3
61	3D	Keyboard F4
62	3E	Keyboard F5
63	3F	Keyboard F6
64	40	Keyboard F7
65	41	Keyboard F8
66	42	Keyboard F9
67	43	Keyboard F10
68	44	Keyboard F11
69	45	Keyboard F12
70	46	Keyboard PrintScreen
71	47	Keyboard Scroll Lock

72	48	Keyboard Pause
73	49	Keyboard Insert
74	4A	Keyboard Home
75	4B	Keyboard PageUp
76	4C	Keyboard Delete Forward
77	4D	Keyboard End
78	4E	Keyboard PageDown
79	4F	Keyboard RightArrow
80	50	Keyboard LeftArrow
81	51	Keyboard DownArrow
82	52	Keyboard UpArrow
83	53	Keypad Num Lock and Clear1
84	54	Keypad /1
85	55	Keypad *
86	56	Keypad -
87	57	Keypad +
88	58	Keypad ENTER
89	59	Keypad 1 and End
90	5A	Keypad 2 and Down Arrow
91	5B	Keypad 3 and PageDn
92	5C	Keypad 4 and Left Arrow
93	5D	Keypad 5
94	5E	Keypad 6 and Right Arrow
95	5F	Keypad 7 and Home
96	60	Keypad 8 and Up Arrow
97	61	Keypad 9 and PageUp
98	62	Keypad 0 and Insert
99	63	Keypad . and Delete
100	64	Keyboard Non-US \ and
101	65	Keyboard Application