AVR-DOS File System

Top Previous Next

AVR-DOS is a Disk Operating System (DOS) for Atmel AVR microcontroller.
The AVR-DOS file system is written by Josef Franz Vögel. He can be contacted via the BASCOM forum.
Josef has put a lot of effort in writing and especially testing the routines.

Topics of AVR-DOS File System:

Introduction

AVR-DOS provide the needed libraries to handle:
- The file system like open and/or create a file, send to or read from a file (Binary files and ASCII files)
- Interface functions (drivers) for Compact Flash, hard disk, SD-Cards, SDHC (also microSD or microSDHC). See SD and SDHC pinout below.

See also: New CF-Card Drivers, Elektor CF-Interface

The Filesystem works with:
- FAT16 formatted partitions
- FAT32 formatted partitions
- Short file name (8.3)
- Files with a long file name can be accessed by their short file name alias
- Files in Root Directory. The root dir can store 512 files. Take in mind that when you use long file names, less filenames can be stored.
- Files in Root directory and sub directories
- LBA mode (Logical block addressing) which is a linear addressing scheme where blocks are located by an integer index.

SD-card is a further development of the former MMC (Multi Media Card).
FAT = File Allocation Table and is the name of the file system architecture (FAT16 means 16-Bit version of FAT).

A SD or SDHC card is working at 2.7V ... 3.6V so for ATMEGA running at 5V you need a voltage converter or voltage divider. ATXMEGA are running at 2.7V ... 3.6V anyway so you can connect the sd-card direct to the ATXMEGA pin's.

 It is very important to use a proper level converter when using high clock rates (above 8 MHz). When using a FET/resistor as a level converter make sure there is enough pull up for a proper clock pulse.

Everything is written in Assembler to ensure a fast and compact code.
The intention in developing the DOS – file system was to keep close to the equivalent VB functions.

 Note that it is not permitted to use the AVR-DOS file system for commercial applications without the purchase of a license. A license comes with the ASM source. You can buy a user license that is suited for most private users.
When you develop a commercial product with AVR-DOS you need the company license. The ASM source is shipped with both licenses.

 Josef nor MCS Electronics can be held responsible for any damage or data loss of your memory cards or disk drives.

FAT16-FAT32

In the root-directory of a FAT16, you have a maximum of 512 directory entries. This limit is defined in the Partition Boot sector. In a FAT16 subdirectory there is a limit of 65535 (2^16 - 1) entries. This Limit depends of the type of the directory entry pointer used in AVR-DOS and can not be increased.

On a FAT32 Partition you have in all kind of directories (Root and Sub) the limit of 65535 entries like the FAT16 Subdirectory.

Please take into account, that long-File-Name Entries will use more than one Directory-Entry depending on the length of the file-name.
So if you use a card with files created on a PC, there a normally more Directory Entries used, than the count of file names.

Important Steps to configure AVR-DOS

1. Driver interface Library (select one of the following):
   For compactFlash:
```
$include "Config_CompactFlash_ElektorIF_M128.bas"
$include "Config_CompactFlash_M128.bas"
```
   For Hard Drives:
```
$include "Config_HardDisk.bas"
```
   For SD-Cards:
```
$include "Config_MMC.bas"
```
   For SD-cards and SDHC cards (works also with ATXMEGA !):
```
$include "config_MMCSD_HC.inc"
```

2. After calling the Driver interface library you need check the Error Byte which is **Gbdriveerror** and which is output of the function DRIVEINIT(). If the output is 0 (no error) you can include the AVR-DOS configuration file. Otherwise you should output the error number.

```
If Gbdriveerror = 0 Then
 $include "Config_AVR-DOS.inc"
End If
```

3. In case of **Gbdriveerror = 0** (No Error) you can Initialize the file system with INITFILESYSTEM(1) where 1 is the partition number. For the Error Output var you need to dim a byte variable like `Dim Btemp1 As Byte` wbefore you call the Initfilesystem.

```
Btemp1 = Initfilesystem(1)
```

   With Btemp1 = 0 (no error) the **Filesystem is successfully initialized** and you can use all

other AVR-DOS functions like open, close, read and write.
Functions like PUT, GET, SEEK-Set only work when the file is opened in binary mode for example: `Open "test.bin" For Binary As #2`

When you want change (ejecting from the card socket) the SD-card (during the AVR is running other code than AVR-DOS) you need to call DRIVEINIT() and INITFILESYSTEM(1) again in order to reset the AVR-Hardware (PORTs, PINs) attached to the Drive,reset the Drive again and initialize the file system again.

 When you include a Constant named SHIELD like : CONST SHIELD=1 , the CS line is kept active which is required for some W5100/W5200 shields.

Requirements:

- Software: appr. 2K-Word Code-Space (4000 Bytes in flash)
- SRAM: 561 Bytes for File system Info and DIR-Handle buffer
- 517 Bytes if FAT is handled in own buffer (for higher speed), otherwise it is handled with the DIR Buffer
- 534 Bytes for each File handle
- This means that a ATMEGA644, ATMEGA128 or ATXMEGA have enough memory for it.
- Even an ATMEGA32 could work but you really need to know what you do and you need to fully understand the settings in `Config_AVR-DOS.BAS` to reduce the amount of SRAM used by AVR-DOS (which will also affect AVR-DOS performance)

For example by setting `Const Cfilehandles = 1` and handling of FAT- and DIR-Buffer in one SRAM buffer with 561 bytes). You will not have much SRAM left anyway for other tasks in the ATMEGA32 and you can not expect maximum performance. $HWSTACK, $SWSTACK and $FRAMESIZE also needs to be set carefully.

```
' Count of file-handles, each file-handle needs 524 Bytes of SRAM
Const Cfilehandles = 1                                    ' [default = 2]

' Handling of FAT-Buffer in SRAM:
' 0 = FAT- and DIR-Buffer is handled in one SRAM buffer with 561 bytes
' 1 = FAT- and DIR-Buffer is handled in separate SRAM buffers with 1078 bytes
' Parameter 1 increased speed of file-handling
Const Csepfathandle = 0                                    ' [default = 1]
```

In the Main.bas you also need a Filename like `Dim File_name As String * 12`

With the above configuration and with the filename there is approximately 500 byte SRAM left in an ATMEGA32 for other tasks. Or in other words AVR-DOS needs at least **1500 Byte SRAM** in this case. To get detailed values compile your AVR-DOS application and open the **Bascom-AVR compiler Report (CTRL+W)** then you see the value with *Space left : 508 Bytes* (then you have 508 Bytes left for other tasks).

Then you can log data with for example:

```
Wait 4

  Open File_name For Append As #100
    Print #100 , "This is what I log to SD-Card !"
  Close #100
```

When you change now `Const Csepfathandle = 1` then you will get an OUT OF SRAM space message from the compiler with an ATMEGA32 which indicates that this will not work with an ATMEGA32.

- Other chips have too little internal memory. You could use XRAM memory too to extend the RAM.
- SPI Interface for SD and SDHC cards (can be used in hardware and software SPI mode where

hardware SPI mode is faster)

To get started there are Examples in the ...BASCOM-AVR\SAMPLES\avrdos folder.

Steps to get started with an ATMEGA (and with MMC.lib):

The MMC.lib is for SD-Cards (Standard SD-Cards usually up to 2Gbyte and not for SDHC cards)

1. Open **Test_DOS_Drive.bas**
2. Add $HWSTACK, $SWSTACK and $FRAMESIZE
3. Add the hardware driver you want to use (for example for SD-Card this is `$include "Config_MMC.bas"`)
4. Open the `Config_MMC.bas` file and configure the SPI interface (hardware or software SPI and which Pin's for example for SPI chip select should be used. `Config_MMC.bas` will call the **MMC.lib** library which is located in the ...BASCOM-AVR\LIB folder.
5. Then you will find in **Test_DOS_Drive.bas** the Include AVR-DOS Configuration and library (`$include "Config_AVR-DOS.BAS"`). `Config_AVR-DOS.BAS` can be also found in ...BASCOM-AVR\SAMPLES \avrdos folder.
6. In `Config_AVR-DOS.BAS` you can change the AVR-DOS user settings like the number of file handles or if AT- and DIR-Buffer is handled in one SRAM buffer or in different SRAM buffer. With this settings you can balance between SRAM space used and speed/performance of AVR-DOS.

File System Configuration in **CONFIG_AVR-DOS.BAS**

cFileHandles:	Count of File handles: for each file opened at same time, a file handle buffer of 534 Bytes is needed
cSepFATHandle:	For higher speed in handling file operations the FAT info can be stored in a own buffer, which needs additional 517 Bytes. Assign Constant cSepFATHandle with 1, if wanted, otherwise with 0.

7. `Config_AVR-DOS.BAS` will call **AVR-DOS.Lbx** library which is located in the ...BASCOM-AVR\LIB folder.
8. Compile, flash and run **Test_DOS_Drive.bas**

Files used in the **Test_DOS_Drive.bas** example:

```
'  +-----------------------------------------------+
'  |               Test_DOS_Drive.bas              |      Main
'  +-----------------------------------------------+
'           |                         |
'  +--------------------+     +----------------------+
'  |   config_MMC.bas   |     | Config_AVR-DOS.bas   |      Include Files
'  +--------------------+     +----------------------+
'           |                         |
'  +--------------------+     +----------------------+
'  |       MMC.lib      |     |     AVR-DOS.Lbx      |      Libraries
'  +--------------------+     +----------------------+
```

Getting started with an ATMEGA and ATXMEGA with MMCSD_HC.LIB:

The **mmcsd_hc.lib** can be found in the ...BASCOM-AVR\LIB folder.

This library support:

- SD-Cards (also known as SDSC Cards = Secure Digital Standard-Capacity usually up to 2 GByte (also microSD)
- SDHC cards (Secure Digital High Capacity) cards start at 2Gbyte up to 32GByte. You can also use

micro SDHC cards.
- It works with ATMEGA and ATXMEGA chips.
- See also : MMCSD_HC.LIB

See ATXMEGA example program below.

Memory Usage of DOS – File System:

1. General File System information (need 35 Byte in SRAM)

Variable Name	Type	Usage
gbDOSError	Byte	holds DOS Error of last file handling routine
gbFileSystem	Byte	File System Code from Master Boot Record
glFATFirstSector	Long	Number of first Sector of FAT Area on the Card
gbNumberOfFATs	Byte	Count of FAT copies
gwSectorsPerFat	Word	Count of Sectors per FAT
glRootFirstSector	Long	Number of first Sector of Root Area on the Card
gwRootEntries	Word	Count of Root Entries
glDataFirstSector	Long	Number of first Sector of Data Area on the Card
gbSectorsPerCluster	Byte	Count of Sectors per Cluster
gwMaxClusterNumber	Word	Highest usable Cluster number
gwLastSearchedCluster	Word	Last cluster number found as free
gwFreeDirEntry	Word	Last directory entry number found as free
glFS_Temp1	Long	temporary Long variable for file system
gsTempFileName	String * 11	temporary String for converting file names

2. Directory (need 559 Byte in SRAM)

Variable Name	Type	Usage
gwDirRootEntry	Word	number of last handled root entry
glDirSectorNumber	Long	Number of current loaded Sector
gbDirBufferStatus	Byte	Buffer Status
gbDirBuffer	Byte (512)	Buffer for directory Sector

3. FAT (need 517 Byte in SRAM)
FAT Buffer is only allocated if the constant: cSepFATHandle = 1

Variable Name	Type	Usage
glFATSectorNumber	Long	Number of current loaded FAT sector
gbFATBufferStatus	Byte	Buffer status
gbFATBuffer	Byte(512)	buffer for FAT sector

4. File handling

Each file handle has a block of 534 Bytes in the variable abFileHandle which is a byte-array of size

(534 * cFileHandles)

Variable Name	Type	Usage
FileNumber	Byte	File number for identification of the file in I/O operations to the opened file
FileMode	Byte	File open mode
FileRootEntry	Word	Number of root entry
FileFirstCluster	Word	First cluster
FATCluster	Word	cluster of current loaded sector
FileSize	Long	file size in bytes
FilePosition	Long	file pointer (next read/write) 0-based
FileSectorNumber	Long	number of current loaded sector
FileBufferStatus	Byte	buffer Status
FileBuffer	Byte(512)	buffer for the file sector
SectorTerminator	Byte	additional 00 Byte (string terminator) for direct reading ASCII files from the buffer

Error Codes:

Code	Compiler – Alias	Remark
0	cpNoError	No Error
1	cpEndOfFile	Attempt behind End of File
17	cpNoMBR	Sector 0 on Card is not a Master Boot Record
18	cpNoPBR	No Partition Sector
19	cpFileSystemNotSupported	Only FAT16 File system is supported
20	cpSectorSizeNotSupported	Only sector size of 512 Bytes is supported
21	cpSectorsPerClusterNotSupported	Only 1, 2, 4, 8, 16, 32, 64 Sectors per Cluster is supported. This are values of normal formatted partitions. Exotic sizes, which are not power of 2 are not supported
22	Cpcountofclustersnotsupported	The number of clusters is not supported
33	cpNoNextCluster	Error in file cluster chain
34	cpNoFreeCluster	No free cluster to allocate (Disk full)
35	cpClusterError	Error in file cluster chain
49	cpNoFreeDirEntry	Directory full
50	cpFileExist	File exists
51	Cpfiledeletenotallowed	File may not be deleted
52	Cpsubdirectorynotempty	Sub directory not empty.You can not delete sub folders which contain files
53	Cpsubdirectoryerror	Sub directory error
54	Cpnotasubdirectory	
65	cpNoFreeFileNumber	No free file number available, only theoretical error, if 255 file handles in use
66	cpFileNotFound	File not found
67	cpFileNumberNotFound	No file handle with such file number
68	cpFileOpenNoHandle	All file handles occupied
69	cpFileOpenHandleInUse	File handle number in use, can't create a new file handle with same file number
70	cpFileOpenShareConflict	Tried to open a file in read and write modus in two file handles

71	cpFileInUse	Can't delete file, which is in use
72	cpFileReadOnly	Can't open a read only file for writing
73	cpFileNoWildCardAllowed	No wildcard allowed in this function
74	Cpfilenumberinvalid	Invalid file number
97	cpFilePositionError	
98	cpFileAccessError	function not allowed in this file open mode
99	cpInvalidFilePosition	new file position pointer is invalid (minus or 0)
100	cpFileSizeToGreat	File size to great for function BLoad
&HC0	Cpdrivererrorstart	

Buffer Status: Bit definitions of Buffer Status Byte (Directory, FAT and File)

Bit	DIR	FAT	File	Compiler Alias	Remark
0 (LSB)			●	dBOF	Bottom of File (not yet supported)
1			●	dEOF	End of File
2			●	dEOFinSector	End of File in this sector (last sector)
3	●	●	●	dWritePending	Something was written to sector, it must be saved to Card, before loading next sector
4		●		dFATSector	This is an FAT Sector, at writing to Card, Number of FAT copies must be checked and copy updated if necessary
5			●	dFileEmpty	File is empty, no sector (Cluster) is allocated in FAT to this file

Validity of the file I/O operations regarding the opening modes

	Open mode			
Action	**Input**	**Output**	**Append**	**Binary**
Attr	●	●	●	●
Close	●	●	●	●
Put				●
Get				●
LOF	●	●	●	●
LOC	●	●	●	●
EOF	●	1)	1)	●
SEEK	●	●	●	●
SEEK-Set				●
Line Input	●			●
Print		●	●	●
Input	●			●
Write		●	●	●

1) Position pointer is always at End of File

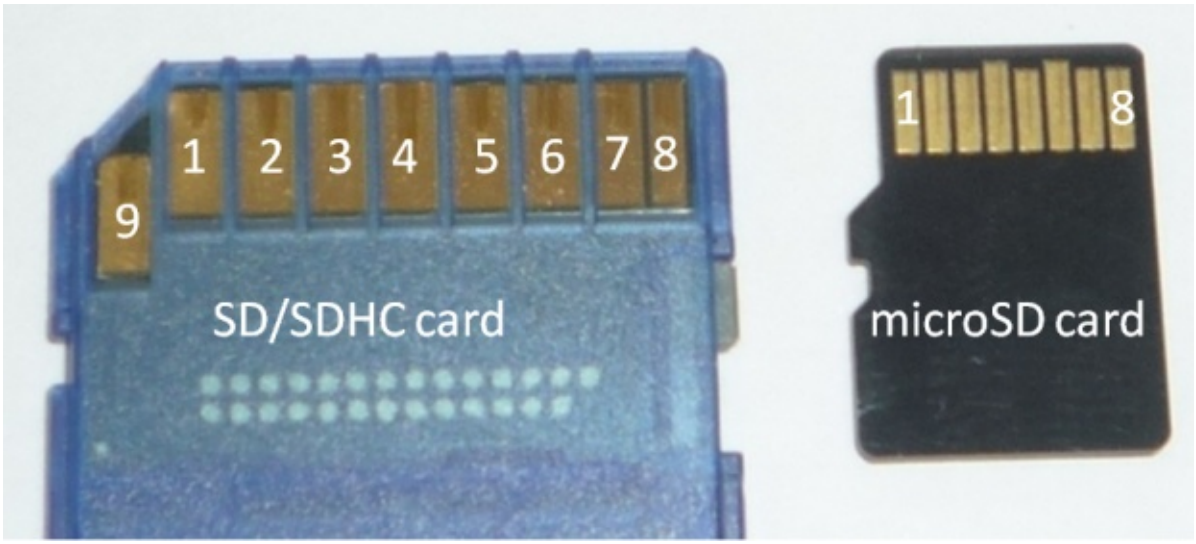
Supported statements and functions:

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT ,FILEDATE , FILETIME , FILEDATETIME , DIR , WRITE , INPUT , FILELEN

SD and SDHC specs and pin-out: (also microSD and microSD pin-out for SPI mode):

SD/SDHC Specs:

- SD and SDHC Cards offer a cost-effective and way to store large amounts of data on a removable memory and is ideal for data logging applications.
- SDHC has a different protocol than SD card with standard Capacity (therefore there was different libraries available at the beginning)
- Standard SD-Cards have a byte addressing. SDHC-Cards have sector-addressing like hard-disks and CF-Cards. One Sector is a portion of 512Bytes. SD cards and SDHC cards also have differences in the protocol at initializing the card, which can be used to check, which kind of card is inserted.
- SD Card operating range: 2.7V...3.6V. So you need a voltage level converter to connect a 5V micro to a SD-card.
- SD cards can be controlled by the six line SD card interface containing the signals: CMD,CLK,DAT0~DAT3 however this is not supported with AVR-DOS.
- AVR-DOS support the SPI interface which can be easily used with the hardware SPI interface of ATMEGA and ATXMEGA. (Software SPI is also supported).
- The SPI mode is active if the CS signal is asserted (negative) during the reception of the reset command (CMD0) which will be automatically handled by AVR-DOS
- The advantage of the SPI mode is reducing the host design in effort.
- With the Chip Select you can also connect several SPI slaves to one SPI interface
- Endurance: Usually SD or SDHC cards can handle typical up to 100,000 writes for each sector. Reading a logical sector is unlimited. Please take care when writing to SD cards in a loop.
- A typical SD Card current consumption should be between 50mA .... 80mA but should not exceed 200mA



Picture: Backside of SD/SDHC card and microSD card

SD/SDHC card pin out:

Pin #	Description for SPI mode	Connect to Pin on ATMEGA128	Connect to Pin on ATXMEGA128A1
1	Chip Select (SS) (Active low)	SS (PortB 0) (Active low)	SS (example for SPIC) PortC 4 (Active low)
2	DI (Data In)	MOSI (PortB 2)	MOSI (example for SPIC) PortC 5
3	GND	GND	GND
4	Vdd (Supply Voltage)	Supply Voltage (2.7V...3.6V)	Supply Voltage (2.7V...3.6V)
5	Clock	SCK (PortB 1)	SCK (example for SPIC) PortC 7
6	GND	GND	GND
7	D0 (Data Out)	MISO (PortB 3)	MISO (example for SPIC) PortC 6
8	Reserved	- - -	- - -
9	Reserved	- - -	- - -

Depending on the used SD-card (or microSD) socket you can also detect if the card is inserted or ejected (for this you need an additional pin on the micro).
In some cases it is best practise to spend another pin able to switch on and off the power to the SD-card socket (e.g. over a transistor or FET). In this case you can cycle power from the AVR when the sd-card controller hangs.
It is also best practise in some cases when you open a file for append, write the data to it and close it right after this so there is no open file where data could be corrupted by an undefined external event.

microSD card pin out (same as microSDHC pin-out):

Pin #	microSD Description for SPI mode
1	Reserved
2	Chip Select (SS)
3	DI (Data In)
4	Vdd (Supply Voltage)
5	Clock
6	GND
7	DO (Data Out)
8	Reserved

Example 1 for getting started with an ATMEGA and ATXMEGA with MMCSD_HC.LIB:

```
'-----------------------------------------------------------------------------
' Filename:         XMEGA_AVR-DOS_SDHC.BAS
' Library needed:   MMCSD_HC.LIB --> Place MMCSD_HC.LIB in the LIB-Path of BASCOM-AVR installation
'                   MMCSD_HC.LIB will be called from config_MMCSD_HC.inc
'                   AVR-DOS.Lbx
' Include file:     config_MMCSD_HC.inc  (will be called from XMEGA_AVR-DOS_SDHC.BAS)
' Used ATXMEGA:     ATXMEGA128A1
' Used SPI Port:    Port D (you can also use Software SPI)
'-----------------------------------------------------------------------------
'
' File Structure:
'
'  +-----------------------------------------------+
'  |             XMEGA_AVR-DOS_SDHC.BAS            |      Main
'  +-----------------------------------------------+
'           |                         |
'  +--------------------+     +----------------------+
'  | config_MMCSD_HC.inc|     | Config_AVR-DOS.inc   |      Include Files
'  +--------------------+     +----------------------+
'           |                         |
'  +--------------------+     +----------------------+
'  |    MMCSD_HC.LIB    |     |     AVR-DOS.Lbx      |      Libraries
'  +--------------------+     +----------------------+
'
```

```
'
' Terminal output of following example (with hardware SPI over Port.D):
'
' Used SD-Card: 4GByte SDHC Card
'
'
'(

---Example for using a SDHC-Card with AVR-DOS and XMEGA---
Starting... SDHC with ATXMEGA....

SD Card Type = SDHC Spec. 2.0 or later

Init File System ...  OK  --> Btemp1= 0 / Gbdriveerror = 0
Filesystem = 6
FAT Start Sector: 8196
Root Start Sector: 8688
Data First Sector: 8720
Max. Cluster Nummber: 62794
Sectors per Cluster: 128
Root Entries: 512
Sectors per FAT: 246
Number of FATs: 2


Write to file done !
File length = 46
This is my 1 first Text to File with XMEGA !
write to file
Total bytes written: 10200
Write and Readback test done !
Dir function demo
LOGGER.TXT  01\01\01  01:00:00  3120
MY_FILE.TXT  01\01\01  01:00:00  46
TEST.TXT  01\01\01  01:00:00  10200

Diskfree = 4018560
Disksize = 4018752

')


$regfile = "xm128a1def.dat"
$crystal = 32000000                                          '32MHz
$hwstack = 128
$swstack = 128
$framesize = 128


Config Osc = Disabled , 32mhzosc = Enabled                  '32MHz
Config Sysclock = 32mhz                                      '32MHz
Config Priority = Static , Vector = Application , Lo = Enabled       'config interrupts
Enable Interrupts

'=====[ Serial Interface to PC = COM5 ]=======================================
Config Com5 = 57600 , Mode = Asynchroneous , Parity = None , Stopbits = 1 , Databits = 8
Open "COM5:" For Binary As #2
Waitms 1

Print #2 ,
Print #2 , "---Example for using a SDHC-Card with AVR-DOS and XMEGA---"

'=====[ Global Vars ]=========================================================
Dim Btemp1 As Byte                                           ' Needed for Fat Drivers
Dim Input_string As String * 100
Dim Output_string As String * 100
Dim File_handle As Byte
Dim File_name As String * 14
Dim X As Long



Print #2 , "Starting... SDHC with ATXMEGA...."
Print #2 ,

'-------------------------------------------------------------------------------

'=====[ Includes ]============================================================


$include "config_MMCSD_HC.inc"
```

```
Print #2 , "SD Card Type = " ;
Select Case Mmcsd_cardtype
Case 0 : Print #2 , "can't init the Card"
Case 1 : Print #2 , "MMC"
Case 2 : Print #2 , "SDSC Spec. 1.x "
Case 4 : Print #2 , "SDSC Spec. 2.0 or later"
Case 12 : Print #2 , "SDHC Spec. 2.0 or later"
End Select

Print #2 ,


If Gbdriveerror = 0 Then                                    'from.... Gbdriveerror = Driveinit()
  $include "Config_AVR-DOS.inc"                            ' Include AVR-DOS Configuration and library


  Print #2 , "Init File System ... " ;
  Btemp1 = Initfilesystem(1)                                ' Reads the Master boot record and the
partition boot record (Sector) from the flash card and initializes the file system
                                                           '1 = Partitionnumber
  If Btemp1 <> 0 Then
    Print #2 , "Error: " ; Btemp1 ; " at Init file system"
  Else
    Print #2 , " OK  --> Btemp1= " ; Btemp1 ; " / Gbdriveerror = " ; Gbdriveerror
    Print #2 , "Filesystem = " ; Gbfilesystem
    Print #2 , "FAT Start Sector: " ; Glfatfirstsector
    Print #2 , "Root Start Sector: " ; Glrootfirstsector
    Print #2 , "Data First Sector: " ; Gldatafirstsector
    Print #2 , "Max. Cluster Nummber: " ; Glmaxclusternumber
    Print #2 , "Sectors per Cluster: " ; Gbsectorspercluster
    Print #2 , "Root Entries: " ; Gwrootentries
    Print #2 , "Sectors per FAT: " ; Glsectorsperfat
    Print #2 , "Number of FATs: " ; Gbnumberoffats
  End If

  Print #2 ,
  Print #2 ,

    '-----------------------------------------------------------------------
    ' Write Text to file
     File_handle = Freefile()                               ' get a file handle
     File_name = "My_file.txt"
    Open File_name For Output As #file_handle               ' open file for output with file_handle
    ' If the file exist already, the file will be overwritten !
    Print #file_handle , "This is my 1 first Text to File with XMEGA !"
    Close #file_handle

    Print #2 , "Write to file done !"

    '-----------------------------------------------------------------------
    'Now we want to read back the text we wrote to file and print it over Serial Interface
     File_handle = Freefile()
    Open "My_file.txt" For Input As #file_handle           ' we can use a constant for the file too
    Print #2 , "File length = " ; Lof(#file_handle)
    Line Input #file_handle , Input_string                 ' read a line
    Print #2 , Input_string                                 'print the line
    Close #file_handle




      'WRITE TO FILE
    Print #2 , "write to file"
    File_name = "Test.txt"
    Input_string =
"12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890"


    Open File_name For Output As #10

      While X < 10000                                       '10000 * 102 Byte / 100 = 10200 Byte
          Print #10 , Input_string
          X = X + 100
      Wend

    Close #10


     X = Filelen(file_name)
    Print #2 , "Total bytes written: " ; X
```

```
    'READ FROM FILE

    Open File_name For Input As #10
      While Eof(#10) = 0
        Line Input #10 , Output_string                      ' read a line
        If Input_string <> Output_string Then
          Print #2 , "Buffer Error! near byte: " ; X ; "   " ; "[" ; Output_string ; "]"
          Waitms 2000
        End If
    Wend
    Close #10


    Print #2 , "Write and Readback test done !"



    '-----------------------------------------------------------------------
    'Print the file name which was created before
    Print #2 , "Dir function demo"
     Input_string = Dir( "*.*")
    'The first call to the DIR() function must contain a file mask The * means everything.
    ' Print File Names
    While Len(input_string) > 0                             ' if there was a file found
        Print #2 , Input_string ; "  " ; Filedate() ; "  " ; Filetime() ; "  " ; Filelen()
      '   print file , the date the fime was created/changed , the time and the size of the file
        Input_string = Dir()                                ' get next
    Wend

    '-----------------------------------------------------------------------

    Print #2 ,
    Print #2 , "Diskfree = " ; Diskfree()
    Print #2 , "Disksize = " ; Disksize()

End If                                                      'If Gbdriveerror = 0 Then


End                                                          'end program
```

Example 1: Following the Config_MMCSD_HC.INC which is included in the main example program:

```
$nocompile

'-------------------------------------------------------------------------------
'                         Config_MMCSD_HC.INC
'               Config File for MMC/SD/SDHC Flash Cards Driver
'        (c) 2003-2009 , MCS Electronics / Vögel Franz Josef
'-------------------------------------------------------------------------------
' Place MMCSD_HC.LIB in the LIB-Path of BASCOM-AVR installation
'
' you can vary MMC_CS on HW-SPI and all pins on SOFT-SPI, check settings
'
' ========== Start of user definable range ===================================
'
' Declare here you SPI-Mode
' using HW-SPI:     cMMC_Soft = 0
Const Hardware_spi = 0
' not using HW_SPI: cMMC_Soft = 1
Const Software_spi = 1

Const Cmmc_soft = Hardware_spi

#if Cmmc_soft = 0
```

```
' --------- Start of Section for HW-SPI ----------------------------------------

  'Port D of ATXMEGA is used in this example as SPI Interface  to SD-Card

   Portd_pin6ctrl = &B00_011_000                          'Enable Pullup for MISO Pin

  ' Define here Slave Slect (SS) Pin of Hardware SPI
  Config Pind.4 = Output                                  ' define here Pin for CS of MMC/SD Card
  Mmc_cs Alias Portd.4
  Set Mmc_cs

  ' Define here Slave Slect (SS) Pin of Hardware SPI
  Config Pind.4 = Output                                  ' define here Pin of SPI SS
  Spi_ss Alias Portd.4
  Set Spi_ss                                              ' Set SPI-SS to Output and High por Proper work
of



  'FOR XMEGA DEVICES
  #if _xmega = 1
      'SPI Configuration for XMEGA
      'Used Library = $LIB "MMCSD_HC.LIB"


      'Portd.4  SS   --> SD-Card  Slave Select
      'Portd.5  MOSI --> SD-Card  MISO
      'Portd.6  MISO --> SD-Card  MOSI
      'Portd.7  CLK  --> SD-Card  Clock

      Config Spid = Hard , Master = Yes , Mode = 0 , Clockdiv = Clk2 , Data_order = Msb
      Open "SPID" For Binary As #14
      Const _mmc_spi = Spid_ctrl
  #else

  ' HW-SPI is configured to highest Speed
  Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes , Polarity = High , Phase = 1 ,
Clockrate = 4 , Noss = 1
'   Spsr = 1                                    ' Double speed on ATMega128
  Spiinit
  #endif

' --------- End of Section for HW-SPI ------------------------------------------

#else                                                      ' Config here SPI pins, if not using HW SPI

' --------- Start of Section for Soft-SPI --------------------------------------

  ' Chip Select Pin  => Pin 1 of MMC/SD
  Config Pind.4 = Output
  Mmc_cs Alias Portd.4
  Set Mmc_cs

  ' MOSI - Pin  => Pin 2 of MMC/SD
  Config Pind.5 = Output
  Set Pind.5
  Mmc_portmosi Alias Portd
  Bmmc_mosi Alias 5

  ' MISO - Pin  => Pin 7 of MMC/SD
  Config Pind.6 = Input
  Mmc_portmiso Alias Pind
  Bmmc_miso Alias 6

  ' SCK - Pin  => Pin 1 of MMC/SD
  Config Pind.7 = Output
  Set Pind.7
  Mmc_portsck Alias Portd
  Bmmc_sck Alias 7

' --------- End of Section for Soft-SPI ----------------------------------------

#endif

' ========== End of user definable range ========================================


'==== Variables For Application ================================================
Dim Mmcsd_cardtype As Byte                                 ' Information about the type of the Card
'   0 can't init the Card
```

```
'   1 MMC
'   2 SDSC Spec. 1.x
'   4 SDSC Spec. 2.0 or later
'  12 SDHC Spec. 2.0 or later

Dim Gbdriveerror As Byte                                ' General Driver Error register
' Values see Error-Codes
'=============================================================================



' ==== Variables for Debug ===================================================
' You can remove remarks(') if you want check this variables in your application
Dim Gbdrivestatusreg As Byte                            ' Driver save here Card response
' Dim gbDriveErrorReg as Byte at GbdriveStatusReg overlay     '
' Dim gbDriveLastCommand as Byte                          ' Driver save here Last Command to Card
Dim Gbdrivedebug As Byte
' Dim MMCSD_Try As Byte                                    ' how often driver tried to initialized the
card
'=============================================================================


'==== Driver internal variables ==============================================
' You can remove remarks(') if you want check this variables in your application
' Dim _mmcsd_timer1 As Word
' Dim _mmcsd_timer2 As Word
'=============================================================================



' Error-Codes
Const Cperrdrivenotpresent = &HE0
Const Cperrdrivenotsupported = &HE1
Const Cperrdrivenotinitialized = &HE2

Const Cperrdrivecmdnotaccepted = &HE6
Const Cperrdrivenodata = &HE7

Const Cperrdriveinit1 = &HE9
Const Cperrdriveinit2 = &HEA
Const Cperrdriveinit3 = &HEB
Const Cperrdriveinit4 = &HEC
Const Cperrdriveinit5 = &HED
Const Cperrdriveinit6 = &HEE

Const Cperrdriveread1 = &HF1
Const Cperrdriveread2 = &HF2

Const Cperrdrivewrite1 = &HF5
Const Cperrdrivewrite2 = &HF6
Const Cperrdrivewrite3 = &HF7
Const Cperrdrivewrite4 = &HF8



$lib "MMCSD_HC.LIB"
$external _mmc
' Init the Card
Gbdriveerror = Driveinit()


' you can remark/remove following two Code-lines, if you dont't use MMCSD_GetSize()
$external Mmcsd_getsize
Declare Function Mmcsd_getsize() As Long


' you can remark/remove following two Code-lines, if you dont't use MMCSD_GetCSD()
' write result of function to an array of 16 Bytes
$external Mmcsd_getcsd
Declare Function Mmcsd_getcsd() As Byte


' you can remark/remove following two Code-lines, if you dont't use MMCSD_GetCID()
' write result of function to an array of 16 Bytes
$external Mmcsd_getcid
Declare Function Mmcsd_getcid() As Byte


' you can remark/remove following two Code-lines, if you dont't use MMCSD_GetOCR()
' write result of function to an array of 4 Bytes
$external Mmcsd_getocr
```

```
Declare Function Mmcsd_getocr() As Byte


' you can remark/remove following two Code-lines, if you dont't use MMCSD_GetSDStat
' write result of function to an array of 64 Bytes
$external Sd_getsd_status
Declare Function Sd_getsd_status() As Byte

' check the usage of the above functions in the sample MMCSD_Analysis.bas
' check also the MMC and SD Specification for the content of the registers CSD, CID, OCR and SDStat
```

Example 1: Following the Config_AVR-DOS.inc which is included in the main example program:

```
$nocompile
' Config File-System for Version 5.5:

' === User Settings ============================================================

' Count of file-handles, each file-handle needs 524 Bytes of SRAM
Const Cfilehandles = 2                                      ' [default = 2]

' Handling of FAT-Buffer in SRAM:
' 0 = FAT- and DIR-Buffer is handled in one SRAM buffer with 561 bytes
' 1 = FAT- and DIR-Buffer is handled in separate SRAM buffers with 1078 bytes
' Parameter 1 increased speed of file-handling
Const Csepfathandle = 1                                      ' [default = 1]

' Handling of pending FAT and Directory information of open files
' 0 = FAT and Directory Information is updated every time a data sector of the file is updated
' 1 = FAT and Directory Information is only updated at FLUSH and SAVE command
' Parameter 1 increases writing speed of data significantly
Const Cfatdirsaveatend = 1                                  ' [default = 1]


' Surrounding String with Quotation Marks at the Command WRITE
' 0 = No Surrounding of strings with quotation.marks
' 1 = Surrounding of strings with quotation.marks (f.E. "Text")
Const Ctextquotationmarks = 1                                ' [default = 1]


' Write second FAT. Windows accepts a not updated second FAT
' PC-Command: chkdsk /f corrects the second FAT, it overwrites the
' second FAT with the first FAT
' set this parameter to 0 for high speed continuing saving data
' 0 = Second FAT is not updated
' 1 = Second FAT is updated if exist
Const Cfatsecondupdate = 1                                  ' [default = 1]


' Character to separate ASCII Values in WRITE - statement (and INPUT)
' Normally a comma (,) is used. but it can be changed to other values, f.E.
' to TAB (ASCII-Code 9) if EXCEL Files with Tab separated values should be
' written or read. This parameter works for WRITE and INPUT
' Parameter value is the ASSCII-Code of the separator
' 44 = comma [default]
' 9 = TAB                                                   ' [default = 44]
Const Cvariableseparator = 44



' === End of User Setting ======================================================


' === Variables for AVR-DOS ====================================================

' FileSystem Basis Informationen
```

```
Dim Gldrivesectors As Long
Dim Gbdoserror As Byte

' Master Boot Record
Dim Gbfilesystem As Byte
' Partition Boot Record
Dim Gbfilesystemstatus As Byte
Dim Glfatfirstsector As Long
Dim Gbnumberoffats As Byte
Dim Glsectorsperfat As Long
Dim Glrootfirstsector As Long
Dim Gwrootentries As Word
Dim Gldatafirstsector As Long
Dim Gbsectorspercluster As Byte
Dim Glmaxclusternumber As Long
Dim Gllastsearchedcluster As Long

' Additional info
Dim Glfs_temp1 As Long

' Block für Directory Handling

Dim Gldirfirstsectornumber As Long

Dim Gwfreedirentry As Word
Dim Glfreedirsectornumber As Long

Dim Gsdir0tempfilename As String * 11
Dim Gwdir0entry As Word                                    ' Keep together with next, otherwise change
_DIR
Dim Gldir0sectornumber As Long

Dim Gstempfilename As String * 11
Dim Gwdirentry As Word
Dim Gldirsectornumber As Long
Dim Gbdirbufferstatus As Byte
Dim Gbdirbuffer(512) As Byte
Const C_filesystemsramsize1 = 594
#if Csepfathandle = 1
Dim Glfatsectornumber As Long
Dim Gbfatbufferstatus As Byte
Dim Gbfatbuffer(512) As Byte
Const C_filesystemsramsize2 = 517
#else
Const C_filesystemsramsize2 = 0
#endif

' File Handle Block
Const Co_filenumber = 0
Const Co_filemode = 1
Const Co_filedirentry = 2 : Const Co_filedirentry_2 = 3
Const Co_filedirsectornumber = 4
Const Co_filefirstcluster = 8
Const Co_filesize = 12
Const Co_fileposition = 16
Const Co_filesectornumber = 20
Const Co_filebufferstatus = 24
Const Co_filebuffer = 25
Const C_filehandlesize = Co_filebuffer + 513               ' incl. one Additional Byte for 00 as string
terminator
                                                           ' for direct text reading from File-buffer
Const C_filehandlesize_m = 65536 - C_filehandlesize        ' for use with add immediate word with subi,
sbci
                                                           ' = minus c_FileHandleSize in Word-Format

Const C_filehandlessize = C_filehandlesize * Cfilehandles


Dim Abfilehandles(c_filehandlessize) As Byte
Const C_filesystemsramsize = C_filesystemsramsize1 + C_filesystemsramsize2 + C_filehandlessize


' End of variables for AVR-DOS ================================================

' Definitions of Constants ====================================================

' Bit definiton for FileSystemStatus

Dfilesystemstatusfat Alias 0 : Const Dfilesystemstatusfat = 0       ' 0 = FAT16, 1 = FAT32
Dfilesystemsubdir Alias 1 : Const Dfilesystemsubdir = 1     ' 0 = Root-Directory, 1 = Sub-Directory
Const Dmfilesystemsubdir =(2 ^ Dfilesystemsubdir)          ' not used yet
```

```
Const Dmfilesystemdirincluster =(2 ^ Dfilesystemstatusfat + 2 ^ Dfilesystemsubdir)       ' not used yet
Dfatsecondupdate Alias 7 : Const Dfatsecondupdate = 7       ' Bit-position for parameter of
                                                            ' Update second FAT in gbFileSystemStatus


' Bit Definitions for BufferStatus (FAT, DIR, File)

Deof Alias 1 : Const Deof = 1 : Const Dmeof =(2 ^ Deof)
Deofinsector Alias 2 : Const Deofinsector = 2 : Const Dmeofinsector =(2 ^ Deofinsector)
Dwritepending Alias 3 : Const Dwritepending = 3 : Const Dmwritepending =(2 ^ Dwritepending)
Dfatsector Alias 4 : Const Dfatsector = 4 : Const Dmfatsector =(2 ^ Dfatsector)       ' For Writing Sector
back (FATNumber times)
Dfileempty Alias 5 : Const Dfileempty = 5 : Const Dmfileempty =(2 ^ Dfileempty)

' New feature for reduce saving
Dfatdirwritepending Alias 6 : Const Dfatdirwritepending = 6 : Const Dmfatdirwritepending =(2 ^
Dfatdirwritepending)
Dfatdirsaveatend Alias 7 : Const Dfatdirsaveatend = 7 : Const Dmfatdirsaveatend =(2 ^ Dfatdirsaveatend)
Dfatdirsaveanyway Alias 0 : Const Dfatdirsaveanyway = 0 : Const Dmfatdirsaveanyway =(2 ^
Dfatdirsaveanyway)



Const Dmeofall =(2 ^ Deof + 2 ^ Deofinsector)
Const Dmeof_empty =(2 ^ Deof + 2 ^ Deofinsector + 2 ^ Dfileempty)


Const Cp_fatbufferinitstatus =(2 ^ Dfatsector)
Const Cp_dirbufferinitstatus = 0


#if Cfatdirsaveatend = 1
Const Cp_filebufferinitstatus =(2 ^ Dfatdirsaveatend)
#else
Const Cp_filebufferinitstatus = 0
#endif



#if Cfatsecondupdate = 0
  Const Cp_fatsecondupdate =(2 ^ Dfatsecondupdate)
#else
  Const Cp_fatsecondupdate = 0
#endif


' Bit definitions for FileMode (Similar to DOS File Attribut)
Dreadonly Alias 0 : Const Dreadonly = 0
'Const cpFileReadOnly = &H21             ' Archiv and read-only Bit set
Const Cpfilewrite = &H20                                    ' Archiv Bit set


' Error Codes

' Group Number is upper nibble of Error-Code
' Group 0 (0-15): No Error or File End Information
Const Cpnoerror = 0
Const Cpendoffile = 1

' Group 1 (17-31): File System Init
Const Cpnombr = 17
Const Cpnopbr = 18
Const Cpfilesystemnotsupported = 19
Const Cpsectorsizenotsupported = 20
Const Cpsectorsperclusternotsupported = 21
Const Cpcountofclustersnotsupported = 22

' Group 2 (32-47): FAT - Error
Const Cpnonextcluster = 33
Const Cpnofreecluster = 34
Const Cpclustererror = 35
' Group 3 (49-63): Directory Error
Const Cpnofreedirentry = 49
Const Cpfileexists = 50
Const Cpfiledeletenotallowed = 51
Const Cpsubdirectorynotempty = 52
Const Cpsubdirectoryerror = 53
Const Cpnotasubdirectory = 54
' Group 4 (65-79): File Handle
Const Cpnofreefilenumber = 65
```

```
Const Cpfilenotfound = 66
Const Cpfilenumbernotfound = 67
Const Cpfileopennohandle = 68
Const Cpfileopenhandleinuse = 69
Const Cpfileopenshareconflict = 70
Const Cpfileinuse = 71
Const Cpfilereadonly = 72
Const Cpfilenowildcardallowed = 73
Const Cpfilenumberinvalid = 74                              ' Zero is not allowed

' Group 7 (97-127): other errors
Const Cpfilepositionerror = 97
Const Cpfileaccesserror = 98
Const Cpinvalidfileposition = 99
Const Cpfilesizetogreat = 100

Const Cpdrivererrorstart = &HC0


' Range 224 to 255 is reserved for Driver

' Other Constants
' File Open Mode /  stored in File-handle return-value of Fileattr(FN#, [1])
Const Cpfileopeninput = 1                                    ' Read
Const Cpfileopenoutput = 2                                  ' Write sequential
'Const cpFileOpenRandom = 4              ' not in use yet
Const Cpfileopenappend = 8                                  ' Write sequential; first set Pointer to end
Const Cpfileopenbinary = 32                                  ' Read and Write; Pointer can be changed by
user


' permission Masks for file access routine regarding to the file open mode
Const Cfilewrite_mode = &B00101010                          ' Binary, Append, Output
Const Cfileread_mode = &B00100001                            ' Binary, Input
Const Cfileseekset_mode = &B00100000                        ' Binary
Const Cfileinputline = &B00100001                            ' Binary, Input
Const Cfileput_mode = &B00100000                            ' Binary
Const Cfileget_mode = &B00100000                            ' Binary

' Directory attributs in FAT16/32
Const Cpfileopenallowed = &B00100001                        ' Read Only and Archiv may be set
Const Cpfiledeleteallowed = &B00100000
Const Cpfilesearchallowed = &B00111101                      ' Do no search hidden Files
' Bit 0 = Read Only
' Bit 1 = Hidden
' Bit 2 = System
' Bit 3 = Volume ID
' Bit 4 = Directory
' Bit 5 = Archiv
' Long File name has Bit 0+1+2+3 set
Dim Lastdosmem As Byte


$lib "AVR-DOS.Lbx"
```

- - - END of EXAMPLE 1 - - -

Example 2: SD and SDHC Card Analysis Example Demo program (Show the Card Capacity and the Card-Register CSD, CID, OCR and SD_Status):

This example uses: $include "Config_MMCSD_HC.bas" which calls following Libary: $lib "MMCSD_HC.LIB"
This example is written for ATMEGA but is also working for ATXMEGA devices.

```
'-----------------------------------------------------------------------------
'                         MMCSD_Analysis.BAS
'                         Test MMC / SD Card
'        (c) 2003-2012 , MCS Electronics / Vögel Franz Josef
'-----------------------------------------------------------------------------
' Test MMC / SD Card
' Show the Card Capacity and the Card-Register CSD, CID, OCR and SD_Status
' First you have to init the Card in the File Config_MMCSD_HC.bas with
' $Include "Config_MMCSD_HC.bas"
```

```
' All Card registers are written with the MSB first to the Byte-array
' f.E. CSD(1) contains then MSB (Bit 120-127) of the CSD-Register

$regfile = "M644pdef.dat"
$crystal = 16000000

$hwstack = 100
$swstack = 100
$framesize = 100


$baud = 57600



Config Serialin = Buffered , Size = 20
Config Clock = Soft

Enable Interrupts

Config Date = Dmy , Separator = .
Print "Test_Dos_Drive compiled at " ; Version()
$include "Config_MMCSD_HC.bas"




Dim Xc As Byte                                              ' for Print - counter
Dim Xd As Byte                                              ' for Print - Counter

Print "Start of Card Analysis"
Print "Last Drive-Error-Code = " ; Gbdriveerror
Print "Gbdrivestatusreg =" ; Gbdrivestatusreg

' Check detected Card Type
Select Case Mmcsd_cardtype
  Case 1
    Print "MMC-Card detected"
  Case 2
    Print "SD-Card Spec. 1.x detected"
  Case 4
    Print "SD-Card Spec. 2.0 detected"
  Case 12
    Print "SD-Card Spec. 2.0 High Capacity detected"
  Case Else
    Print "No Card detected"
End Select

If Mmcsd_cardtype > 0 Then

' check the CSD Register

  Dim Csd(16) As Byte
  Print "Get CSD"
  Csd(1) = Mmcsd_getcsd()
  If Gbdriveerror <> 0 Then
    Print "Error at reading CSD"
  Else
    For Xc = 1 To 16
        Print Hex(csd(xc)) ; " " ;
    Next
    Print " "
  End If

' Get the Card Capacity from the CSD Register

  Dim Mmcsd_size As Long
  Print "Get Card Capacity [KB]"
  Mmcsd_size = Mmcsd_getsize()
  If Gbdriveerror <> 0 Then
    Print "Error at reading CSD"
  Else
    Print "Card Capacity = ; " ; Mmcsd_size ; "kb  (1KB=1024 Bytes)"
  End If

' Get the CID Register

  Dim Cid(16) As Byte
  Print "Get CID"
```

```
  Cid(1) = Mmcsd_getcid()
  If Gbdriveerror <> 0 Then
    Print "Error at reading CID"
  Else
    For Xc = 1 To 16
        Print Hex(cid(xc)) ; " " ;
    Next
    Print " "
  End If

' Get the OCR Register

  Dim Ocr(4) As Byte
  Print "Get OCR"
  Ocr(1) = Mmcsd_getocr()
  If Gbdriveerror <> 0 Then
    Print "Error at reading OCR"
  Else
    For Xc = 1 To 4
        Print Hex(ocr(xc)) ; " " ;
    Next
    Print " "
  End If

  If Mmcsd_cardtype > 1 Then

' Get the SD_Status Register on SD-Cards

    Dim Sd_status(64) As Byte
    Print "Get SD_Status"
     Sd_status(1) = Sd_getsd_status()
    If Gbdriveerror <> 0 Then
      Print "Error at reading SD_Status"
    Else
      For Xc = 1 To 64
          Print Hex(sd_status(xc)) ; " " ;
          Xd = Xc Mod 8
          If Xd = 0 Then
            Print " "
          End If
      Next
    End If
  End If
End If


Print "End of Card Analysis"

End
```

INITFILESYSTEM

Top Previous Next

Action
Initialize the file system

Syntax
bErrorCode = **INITFILESYSTEM** (bPartitionNumber)

Remarks

bErrorCode	(Byte) Error Result from Routine, Returns 0 if no Error
bPartitionNumber	(Byte) Partition number on the Flashcard Drive (normally 1)

Reads the Master boot record and the partition boot record (Sector) from the flash card and initializes the file system.
This function must be called before any other file-system function is used.

See also
OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT, AVR-DOS File System

ASM

Calls	_GetFileSystem	
Input	r24: partitionnumber (1-based)	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example
```
Dim bErrorCode as Byte
bErrorCode = InitFileSystem(1)
If bErrorCode > 0 then
   Print "Error: "; bErrorCode
Else
   Print "Filesystem successfully initialized"
End If
```

OPEN

Top Previous Next

Action
Opens a device.

Syntax
OPEN "device" for MODE As #channel
OPEN file FOR MODE as #channel

Remarks

Device	The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device. With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use. COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stop bits. The format for COM1 and COM2 is : COM1: or COM2: There is no speed/baud rate parameter since the default baud rate will be used which is specified with $BAUD or $BAUD1 The format for the software UART is: COMpin:speed,8,N,stopbits[,INVERTED] Where pin is the name of the PORT-pin. Speed must be specified and stop bits can be 1 or 2. 7 bit data or 8 bit data may be used. For parity N, O or E can be used. An optional parameter ,INVERTED can be specified to use inverted RS-232. Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232. For the AVR-DOS file system, Device can also be a string or filename constant like "readme.txt" or sFileName For the Xmega, you can also open SPIC, SPID, SPIE and SPIF for SPI communication. Or for TWI you can use TWIC, TWID, TWIE or TWIF.
MODE	You can use BINARY or RANDOM for COM1 and COM2, but for the software UART pins, you must specify INPUT or OUTPUT. For the AVR-DOS file system, MODE may be INPUT, OUTPUT, APPEND or BINARY.
Channel	The number of the channel to open. Must be a positive constant >0. For the AVR-DOS file system, the channel may be a positive constant or a numeric variable. Note that the AVD-DOS file system uses real file handles. The software UART does not use real file handles. For the Xmega UART, you may use a variable that starts with BUART. This need to be a numeric variable like a byte. Using a variable allows you to use

	the UART dynamic.

UART

The statements that support the device are PRINT , INPUT , INPUTHEX , INKEY and WAITKEY

Every opened device must be closed using the CLOSE #channel statement. Of course, you must use the same channel number.

In DOS the #number is a DOS file number that is passed to low level routines. In BASCOM the channel number is only used to identify the channel but there are no file handles. So opening a channel, will not use a channel. Closing a channel is not needed for UARTS. When you do so, it is ignored. If you OPEN the channel again, you will get an error message.
So use OPEN in the begin of your program, and if you use CLOSE, use it at the end of your program.

What is the difference?

In VB you can close the channel in a subroutine like this:

```
OPEN "com1:" for binary as #1
Call test
Close #1
End

Sub test
Print #1, "test"
End Sub
```

This will work since the file number is a real variable in the OS.
In BASCOM it will not work : the CLOSE must come after the last I/O statement:

```
OPEN "com1:" for binary as #1
Call test
End

Sub test
Print #1, "test"
End Sub
Close #1
```

The INPUT statement in combination with the software UART, will not echo characters back because there is no default associated pin for this.

AVR-DOS

The AVR-DOS file system uses real file handles. This means that the CLOSE statement can be used at any place in your program just as with VB.

There are a few file mode, all inherited from VB/QB. They work exactly the same.

File mode	Description
OUTPUT	Use OUTPUT to create a file, and to write ASCII data to the file. A readme.txt file on your PC is an example of an ASCII file. ASCII files have a trailing CR+LF for each line you print. The PRINT statement is used in combination with OUTPUT mode.

INPUT	This mode is intended to OPEN an ASCII file and to read data only. You can not write data in this mode. The file need to exist, and must contain ASCII data. LINEINPUT can be used to read data from the file.
APPEND	APPEND mode is used on ASCII files and will not erase the file, but will append data to the end of the file. This is useful when you want to log data to a file. Opening in OUTPUT mode would erase the file if it existed.
BINARY	In BINARY mode you have full read and write access to all data in the file. You can open a text file to get binary access, or you can open a binary file such as an image file. GET and PUT can be used with binary files.

 The following information from the author is for advanced users only.

GET/PUT is not supposed to work with INPUT/OUTPUT due to the rules in VB/QBASIC.
In the file CONFIG_AVR-DOS.bas (nearly at the of the file) you will find the constants
' permission Masks for file access routine regarding to the file open mode
Const cFileWrite_Mode = &B00101010 ' Binary, Append, Output
Const cFileRead_Mode = &B00100001 ' Binary, Input
Const cFileSeekSet_Mode = &B00100000 ' Binary
Const cFileInputLine = &B00100001 ' Binary, Input
Const cFilePut_Mode = &B00100000 ' Binary
Const cFileGet_Mode = &B00100000 , Binary

Where you can control, which routines can used in each file open mode. There you can see, that in standard usage GET and PUT is only allowed in BINARY.
Some time ago I wrote the Bootloader with AVR-DOS and I had the problem to keep Flash usage as low as possible. In the Bootloader I had to work with GET to read in the bytes, because the content is no ASCII text. On the other side, if you open a file in INPUT mode, you need less code. So I tested to open the File in input mode and allow to use GET in Input Mode.

I changed:
Const cFileGet_Mode = &B00100001
So GET can work in INPUT too in the BOOTLOADER.

If you switch in the constants cFileGet_Mode the last 0 to a 1, you can use GET in INPUT Open mode to.
With the bootloader.bas I changed the Config_AVR-DOS.bas too. With this changed Config_AVR-DOS.bas GET can used in INPUT, with the standard CONFIG_AVR-DOS not.
This change makes no problem in code, but I think this is only something for experienced AVR-DOS user. Whether he can use GET in INPUT mode depends only on this last bit in the constant cFileGET_Mode in the file Config_AVR-DOS.bas. This bit controls, what can be used in INPUT mode.

Xmega-SPI

The Xmega has 4 SPI interfaces. The channel is used to communicate with the different devices.
And just as with the Xmega UART, you can use the SPI dynamic. When the channel variable starts with BSPI, you can pass a variable channel.
An example you will find at CONFIG SPIx
You can OPEN a SPI device only in BINARY mode.

Xmega-TWI

The Xmega has 4 TWI interfaces. The channel is used to communicate with the different devices.
You can OPEN a TWI device only in BINARY mode. Only constants are allowed for the channel.

See also

CLOSE , CRYSTAL, PRINT, LINE INPUT , LOC , LOF , EOF

Example

```
'-----------------------------------------------------------------------------
'name                     : open.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates software UART
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------

$regfile = "m48def.dat"                                    ' specify the used micro
$crystal = 10000000                                          ' used crystal frequency
$baud = 19200                                                ' use baud rate
$hwstack = 32                                                ' default use 32 for the
hardware stack
$swstack = 10                                                ' default use 10 for the SW
stack
$framesize = 40                                              ' default use 40 for the
frame space


Dim B As Byte

'Optional you can fine tune the calculated bit delay
'Why would you want to do that?
'Because chips that have an internal oscillator may not
'run at the speed specified. This depends on the voltage, temp etc.
'You can either change $CRYSTAL or you can use
'BAUD #1,9610

'In this example file we use the DT006 from www.simmstick.com
'This allows easy testing with the existing serial port
'The MAX232 is fitted for this example.
'Because we use the hardware UART pins we MAY NOT use the hardware UART
'The hardware UART is used when you use PRINT, INPUT or other related statements
'We will use the software UART.
Waitms 100

'open channel for output
Open "comd.1:19200,8,n,1" For Output As #1
Print #1 , "serial output"


'Now open a pin for input
Open "comd.0:19200,8,n,1" For Input As #2
'since there is no relation between the input and output pin
'there is NO ECHO while keys are typed
Print #1 , "Number"
'get a number
Input #2 , B
'print the number
Print #1 , B

'now loop until ESC is pressed
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
'store in byte
 B = Inkey(#2)
'when the value > 0 we got something
If B > 0 Then
    Print #1 , Chr(b)                                      'print the character
```

```
End If
Loop Until B = 27


Close #2
Close #1


'OPTIONAL you may use the HARDWARE UART
'The software UART will not work on the hardware UART pins
'so you must choose other pins
'use normal hardware UART for printing
'Print B


'When you dont want to use a level inverter such as the MAX-232
'You can specify ,INVERTED :
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Now the logic is inverted and there is no need for a level converter
'But the distance of the wires must be shorter with this
End
```

Example XMEGA TWI

```
'-----------------------------------------------------------------
'                  (c) 1995-2010, MCS
'                      xm128-TWI.bas
'  This sample demonstrates the Xmega128A1 TWI
'-----------------------------------------------------------------

$regfile = "xm128a1def.dat"
$crystal = 32000000
$hwstack = 64
$swstack = 40
$framesize = 40

Dim S As String * 20

'first enable the osc of your choice
Config Osc = Enabled , 32mhzosc = Enabled

'configure the systemclock
Config Sysclock = 32mhz , Prescalea = 1 , Prescalebc = 1_1

Dim N As String * 16 , B As Byte
Config Com1 = 19200 , Mode = Asynchroneous , Parity = None , Stopbits = 1 , Databits = 8
Config Input1 = Cr , Echo = Crlf                            ' CR is used for input, we
echo back CR and LF

Open "COM1:" For Binary As #1
'       ^^^^ change from COM1-COM8

Print #1 , "Xmega revision:" ; Mcu_revid                    ' make sure it is 7 or higher
!!! lower revs have many flaws

Const Usechannel = 1


Dim B1 As Byte , B2 As Byte
Dim W As Word At B1 Overlay




Open "twic" For Binary As #4                                ' or use TWID,TWIE oR TWIF
```

```
Config Twi = 100000                                         'CONFIG TWI will ENABLE the
TWI master interface
'you can also use TWIC, TWID, TWIE of TWIF

#if Usechannel = 1
  I2cinit #4
#else
I2cinit
#endif


Do
I2cstart
Waitms 20
I2cwbyte &H70                                               ' slave address write
Waitms 20
I2cwbyte &B10101010                                         ' write command
Waitms 20
I2cwbyte 2
Waitms 20
I2cstop
Print "Error : " ; Err                                     ' show error status

'waitms 50
Print "start"
I2cstart
Print "Error : " ; Err                                     ' show error
I2cwbyte &H71
Print "Error : " ; Err                                     ' show error
I2crbyte B1 , Ack
Print "Error : " ; Err                                     ' show error
I2crbyte B2 , Nack
Print "Error : " ; Err                                     ' show error
I2cstop
Print "received A/D : " ; W ; "-" ; B1 ; "-" ; B2
Waitms 500                                                  'wait a bit
Loop



Dim J As Byte , C As Byte , K As Byte
Dim Twi_start As Byte                                       ' you MUST dim this variable
since it is used by the lib

'determine if we have an i2c slave on the bus
For J = 0 To 200 Step 2
Print J
#if Usechannel = 1
  I2cstart #4
#else
  I2cstart
#endif

  I2cwbyte J
  If Err = 0 Then                                           ' no errors
    Print "FOUND : " ; Hex(j)
    'write some value to the pcf8574A
    #if Usechannel = 1
        I2cwbyte &B1100_0101 , #4
    #else
        I2cwbyte &B1100_0101
    #endif
    Print Err
    Exit For
  End If
  #if Usechannel = 1
      I2cstop #4
```

```
  #else
      I2cstop
  #endif
Next
#if Usechannel = 1
  I2cstop #4
#else
  I2cstop
#endif

#if Usechannel = 1
I2cstart #4
I2cwbyte &H71 , #4                                  'read address
I2crbyte J , Ack , #4
Print Bin(j) ; " err:" ; Err
I2crbyte J , Ack , #4
Print Bin(j) ; " err:" ; Err
I2crbyte J , Nack , #4
Print Bin(j) ; " err:" ; Err
I2cstop #4
#else
I2cstart
I2cwbyte &H71                                        'read address
I2crbyte J , Ack
Print Bin(j) ; " err:" ; Err
I2crbyte J , Ack
Print Bin(j) ; " err:" ; Err
I2crbyte J , Nack
Print Bin(j) ; " err:" ; Err
I2cstop
#endif

'try a transaction
#if Usechannel = 1
I2csend &H70 , 255 , #4                              ' all 1
Waitms 1000
I2csend &H70 , 0 , #4                                'all 0
#else
I2csend &H70 , 255
Waitms 1000
I2csend &H70 , 0
#endif
Print Err


'read transaction
Dim Var As Byte
Var = &B11111111
#if Usechannel = 1
I2creceive &H70 , Var , 1 , 1 , #4                   ' send and receive
Print Bin(var) ; "-" ; Err
I2creceive &H70 , Var , 0 , 1 , #4                   ' just receive
Print Bin(var) ; "-" ; Err
#else
I2creceive &H70 , Var , 1 , 1                        ' send and receive
Print Bin(var) ; "-" ; Err
I2creceive &H70 , Var , 0 , 1                        ' just receive
Print Bin(var) ; "-" ; Err
#endif

End
```

CLOSE

Top Previous Next

Action
Closes an opened device.

Syntax
OPEN "device" for MODE As #channel
CLOSE #channel

Remarks

Device	The default device is COM1 and you don't need to open a channel to use INPUT/OUTPUT on this device. With the implementation of the software UART, the compiler must know to which pin/device you will send/receive the data. So that is why the OPEN statement must be used. It tells the compiler about the pin you use for the serial input or output and the baud rate you want to use. COMB.0:9600,8,N,2 will use PORT B.0 at 9600 baud with 2 stop bits. The format for COM1 is : COM1: Some chips have 2 UARTS. You can use COM2: to open the second HW UART. Other chips might have 4 or 8 UARTS. The format for the software UART is: COMpin:speed,8,N,stop bits[,INVERTED] Where pin is the name of the PORT-pin. Speed must be specified and stop bits can be 1 or 2. An optional parameter ,INVERTED can be specified to use inverted RS-232. Open "COMD.1:9600,8,N,1,INVERTED" For Output As #1 , will use pin PORTD.1 for output with 9600 baud, 1 stop bit and with inverted RS-232.
MODE	You can use BINARY or RANDOM for COM1 and COM2, but for the software UART pins, you must specify INPUT or OUTPUT.
Channel	The number of the channel to open. Must be a positive constant >0.

The statements that support the device are PRINT , INPUT and INPUTHEX , INKEY, WAITKEY.

Every opened device must be closed using the CLOSE #channel statement. Of course, you must use the same channel number.

The best place for the CLOSE statement is at the end of your program.

The INPUT statement in combination with the software UART, will not echo characters back because there is no default associated pin for this.

 For the AVR-DOS file system, you may place the CLOSE at any place in your program. This because the file system supports real file handles.
For the UART, SPI or other devices, you do not need to close the device. Only AVR-DOS needs a CLOSE so the file will be flushed.

See also
OPEN , PRINT

Example

```
'-----------------------------------------------------------------------------------------
'name                     : open.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demonstrates software UART
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------------------

$regfile = "m48def.dat"                                    ' specify the used micro
$crystal = 10000000                                          ' used crystal frequency
$baud = 19200                                                ' use baud rate
$hwstack = 32                                                ' default use 32 for the
hardware stack
$swstack = 10                                                ' default use 10 for the SW
stack
$framesize = 40                                              ' default use 40 for the
frame space


Dim B As Byte

'Optional you can fine tune the calculated bit delay
'Why would you want to do that?
'Because chips that have an internal oscillator may not
'run at the speed specified. This depends on the voltage, temp etc.
'You can either change $CRYSTAL or you can use
'BAUD #1,9610

'In this example file we use the DT006 from www.simmstick.com
'This allows easy testing with the existing serial port
'The MAX232 is fitted for this example.
'Because we use the hardware UART pins we MAY NOT use the hardware UART
'The hardware UART is used when you use PRINT, INPUT or other related statements
'We will use the software UART.
Waitms 100

'open channel for output
Open "comd.1:19200,8,n,1" For Output As #1
Print #1 , "serial output"


'Now open a pin for input
Open "comd.0:19200,8,n,1" For Input As #2
'since there is no relation between the input and output pin
'there is NO ECHO while keys are typed
Print #1 , "Number"
'get a number
Input #2 , B
'print the number
Print #1 , B

'now loop until ESC is pressed
'With INKEY() we can check if there is data available
'To use it with the software UART you must provide the channel
Do
'store in byte
 B = Inkey(#2)
'when the value > 0 we got something
If B > 0 Then
    Print #1 , Chr(b)                                    'print the character
End If
Loop Until B = 27
```

```
Close #2
Close #1


'OPTIONAL you may use the HARDWARE UART
'The software UART will not work on the hardware UART pins
'so you must choose other pins
'use normal hardware UART for printing
'Print B
'When you dont want to use a level inverter such as the MAX-232
'You can specify ,INVERTED :
'Open "comd.0:300,8,n,1,inverted" For Input As #2
'Now the logic is inverted and there is no need for a level converter
'But the distance of the wires must be shorter with this
End
```

FLUSH

Top Previous Next

Action
Write current buffer of File to Card and updates Directory

Syntax
FLUSH #bFileNumber
FLUSH

Remarks

BFileNumber	Filenumber, which identifies an opened file such as #1 or #ff

This function writes all information of an open file, which is not saved yet to the Disk. Normally the Card is updated, if a file will be closed or changed to another sector.

When no file number is specified, all open files will be flushed.
Flush does not need additional buffers. You could use FLUSH to be absolutely sure that data is written to the disk. For example in a data log application which is updated infrequently. A power failure could result in a problem when there would be data in the buffer.

See also
INITFILESYSTEM , OPEN , CLOSE, PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_FileFlush	_FilesAllFlush
Input	r24: filenumber	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example
```
$include "startup.inc"

'open the file in BINARY mode
Open "test.bin" For Binary As #2
Put #2 , B                                      ' write a byte
Put #2 , W                                      ' write a word
Put #2 , L                                      ' write a long
Ltemp = Loc(#2) + 1                      ' get the position of the next byte
Print Ltemp ;" LOC"                     ' store the location of the file pointer
Print Lof(#2);" length of file"
Print Fileattr(#2);" file mode"        ' should be 32 for binary
Put #2 , Sn                               ' write a single
Put #2 , Stxt                               ' write a string

Flush #2                                       ' flush to disk
Close #2
```

PRINT

Top Previous Next

Action
Send output to the UART.
Writes a string to a file.
Writes data to a device.

Syntax
PRINT [#channel ,] var ; " constant"

Remarks

Var	The variable or constant to print.

You can use a semicolon (**;**) to print multiple variables or constants after each other.
When you end a line with a semicolon, no linefeed and carriage return will be added.

The PRINT routine can be used when you have a RS-232 interface on your processor.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator as an output device.
You can also use the build in terminal emulator.
When using RS-485 you can use CONFIG PRINT to set up a pin for the direction.
When printing arrays, you can only print one element at the time. When you need to print the content of a complete array, you need to use PRINTBIN.

PRINT will automatic convert numeric variables into the string representation.
This means that when you have a byte variable named B with the value of 123, the numeric variable is converted into a string "123" and then printed.
In this case, print will print 3 characters or bytes. When you want to print the byte you can use the chr() function : print chr(b);
This will send just one byte to the UART.

You can connect the processors UART (TX/RX pins) to a MAX232, an FTDI232RL, a Bluetooth module or a GPS modem. Always check the logic level vcc of the UART and the device you connect to. Connecting 5V devices to a 3v3 device might damage the 3v3 device.
A serial port can be used to update firmware with a so called boot loader.

AVR-DOS
The AVR-DOS file system also supports PRINT. But in that case, only strings can be written to disk.
When you need to print to the second hardware UART, or to a software UART, you need to specify a channel : PRINT **#1,** "test"
The channel must be opened first before you can print to it. Look at OPEN and CLOSE for more details about the optional channel.
For the first hardware UART, there is no need to use channels. The default for PRINT without a channel specifier, is the first UART.
So : *PRINT " test"* will always use the first hardware UART.

Xmega-SPI
When sending data to the SPI interface, you need to activate the SS pin. Some chips might need an active low, others might need an active high. This will depends on the slave chip.
When you use the SS=AUTO option, the level of SS will be changed automatic. Thus SS is made low, then the data bytes are sent, and finally , SS is made high again.

For SPI, no CRLF will be sent. Thus a trailing **;** is not needed.

SPI Number of Bytes
The compiler will send 1 byte for variable which was dimensioned as a BYTE.
It will send 2 bytes for a WORD/INTEGER, 4 bytes for a LONG/SINGLE and 8 bytes for a DOUBLE.
As with all routines in BASCOM, the least significant Byte will be send first.

When you send a numeric constant, the binary value will be sent : 123 will be send a 1 byte with the value of 123.

If you send an array element, one element will be send.
With an optional parameter you can provide how many bytes must be sent. You must use a comma (,) to specify this parameter.
This because the semi colon (;) is used to send multiple variables.

Sample
```
Dim Tmparray(5) As Byte, Spi_send_byte As Byte, W as Word
Config Spid = Hard, Master = Yes, Mode = 0, Clockdiv = Clk32, Data_order = Msb , Ss = Auto
Open "SPID" For Binary As #12
Print #12, Spi_send_byte; W             ' send ONE BYTE and 2 bytes of W
Print #12, Tmparray(1) , 2                ' send 2 bytes of tmparray, starting at element 1
Print #12, Tmparray(1)              ' send 1 byte
Print #12, Tmparray(3) , 2          ' send 2 bytes starting at index 3
```

```
Print #12, 123; 1000; Tmparray(1), B' send byte with value 123, 2 bytes with value 1000, and 'b' bytes of array
```

See also

INPUT,OPEN , CLOSE , SPC , PRINTBIN , HEX, BIN

Example

```
'-----------------------------------------------------------------------------------------
'name                     : print.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: PRINT, HEX
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------------------

$regfile = "m48def.dat"                                    ' specify the used micro
$crystal = 4000000                                          ' used crystal frequency
$baud = 19200                                               ' use baud rate
$hwstack = 32                                               ' default use 32 for the hardware stack
$swstack = 10                                               ' default use 10 for the SW stack
$framesize = 40                                             ' default use 40 for the frame space

Dim A As Byte , B1 As Byte , C As Integer , S As String * 4
A = 1
Print "print variable a " ; A
Print                                                      ' new line
Print "Text to print."                                      ' constant to print

B1 = 10
Print Hex(b1)                                              ' print in hexa notation
C = &HA000                                                  ' assign value to c%
Print Hex(c)                                               ' print in hex notation
Print C                                                     ' print in decimal notation

C = -32000
Print C
Print Hex(c)
Rem Note That Integers Range From -32767 To 32768

Print "You can also use multiple" _
; "lines using _"
Print "use it for long lines"
'From version 1.11.6.4 :
A = &B1010_0111
Print Bin(a)
S = "1001"
A = Binval(s)
Print A                                                     '9 dec
End
```

LINEINPUT

Top Previous Next

Action
Read a Line from an opened File.

Syntax
LINEINPUT #bFileNumber, sLineText
LINE_INPUT #bFileNumber, sLineText

Remarks

BfileNumber	(Byte) File number, which identifies an opened file
SlineText	(String) A string, which is assigned with the next line from the file.

Only valid for files opened in mode INPUT. Line INPUT works only with strings. It is great for working on text files.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_FileLineInput	
Input	r24: filenumber	X: Pointer to String to be written from file
	r25: Stringlength	
Output	r25: Errorcode	C-Flag: Set on Error

Example
```
'Ok we want to check if the file contains the written lines
Ff = Freefile()' get file handle
Open "test.txt" For Input As #ff         ' we can use a constant for the file too
Print Lof(#ff); " length of file"
Print Fileattr(#ff); " file mode"        ' should be 1 for input
Do
  LineInput #ff , S                        ' read a line
  ' line input is used to read a line of text from a file
  Print S                                          ' print on terminal emulator
Loop Until Eof(ff)<> 0
'The EOF() function returns a non-zero number when the end of the file is reached
'This way we know that there is no more data we can read
Close #ff
```

LOC

Top Previous Next

Action
Returns the position of last read or written Byte of the file

Syntax
lLastReadWritten = **LOC** (#bFileNumber)

Remarks

bFileNumber	(Byte) File number, which identifies an opened file
lLastReadWritten	(Long) Variable, assigned with the Position of last read or written Byte (1-based)

This function returns the position of the last read or written Byte. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError. If the file position pointer is changed with the command SEEK, this function can not be used till the next read/write operation.

This function differs from VB. In VB the byte position is divided by 128.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_FileLoc	
Input	r24: filenumber	X: Pointer to Long-variable, which gets th result
Output	r25: Errorcode	C-Flag: Set on Error

Example
```
' open the file in BINARY mode
Open "test.bin" For Binary As #2
Put #2 , B                                      ' write a byte
Put #2 , W                                      ' write a word
Put #2 , L                                      ' write a long
Ltemp = Loc(#2)+ 1                          ' get the position of the next byte
Print Ltemp ;" LOC"                         ' store the location of the file pointer
Print Lof(#2);" length of file"
Print Fileattr(#2);" file mode"         ' should be 32 for binary
Put #2 , Sn                                  ' write a single
Put #2 , Stxt                                 ' write a string

Flush #2                                        ' flush to disk
Close #2
```

LOF

Top Previous Next

Action
Returns the length of the File in Bytes

Syntax
lFileLength = **LOF** (#bFileNumber)

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
LFileLength	(Long) Variable, which assigned with the Length of the file (1-based)

This function returns the length of an opened file. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_FileLOF	
Input	r24: filenumber	X: Pointer to Long-variable, which gets th result
Output	r25: Errorcode	C-Flag: Set on Error

Example
```
' open the file in BINARY mode
Open "test.bin" For Binary As #2
Put #2 , B                                      ' write a byte
Put #2 , W                                      ' write a word
Put #2 , L                                      ' write a long
Ltemp = Loc(#2)+ 1                          ' get the position of the next byte
Print Ltemp ;" LOC"                         ' store the location of the file pointer
Print Lof(#2);" length of file"
Print Fileattr(#2);" file mode"         ' should be 32 for binary
Put #2 , Sn                                  ' write a single
Put #2 , Stxt                                 ' write a string

Flush #2                                        ' flush to disk
Close #2
```

EOF

Top Previous Next

Action
Returns the End of File Status.

Syntax
bFileEOFStatus = **EOF**(#bFileNumber)

Remarks

bFileEOFStatus	(Byte) A Byte Variable, which assigned with the EOF Status
bFileNumber	(Byte) Number of the opened file

This functions returns information about the End of File Status

Return value	Status
0	NOT EOF
255	EOF

In case of an error (invalid file number) 255 (EOF) is returned too.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_FileEOF	
Input	r24: Filenumber	
Output	r24: EOF Status	r25: Error code
	C-Flag: Set on Error	

Partial Example
```
Ff =Freefile()' get file handle
Open "test.txt" For Input As #ff         ' we can use a constant for the file too
Print Lof(#ff); " length of file"
Print Fileattr(#ff); " file mode"        ' should be 1 for input
Do
  LineInput #ff , S                        ' read a line
  ' line input is used to read a line of text from a file
  Print S                                          ' print on terminal emulator
Loop Until Eof(#ff)<> 0
'The EOF() function returns a non-zero number when the end of the file is reached
'This way we know that there is no more data we can read
Close #ff
```

FREEFILE

Top Previous Next

Action
Returns a free Filenumber.

Syntax
bFileNumber = **FREEFILE**()

Remarks

bFileNumber	A byte variable , which can be used for opening next file

This function gives you a free file number, which can be used for file – opening statements. In contrast to VB this file numbers start with 128 and goes up to 255. Use range 1 to 127 for user defined file numbers to avoid file number conflicts with the system numbers from FreeFile()

This function is implemented for compatility with VB.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_GetFreeFileNumber	
Input	none	
Output	r24: Filenumber	r25: Errorcode
	C-Flag: Set on Error	

Partial Example
```
Ff =Freefile()                                  ' get file handle
Open"test.txt" For Input As #ff         ' we can use a constant for the file too
Print Lof(#ff);" length of file"
Print Fileattr(#ff);" file mode"         ' should be 1 for input
Do
   LineInput #ff , S                            ' read a line
  ' line input is used to read a line of text from a file
  Print S ' print on terminal emulator
Loop UntilEof(ff)<> 0
'The EOF() function returns a non-zero number when the end of the file is reached
'This way we know that there is no more data we can read
Close #ff
```

FILEATTR

Top Previous Next

Action
Returns the file open mode.

Syntax
bFileAttribut = **FILEATTR**(bFileNumber)

Remarks

bFileAttribut	(Byte) File open mode, See table
bFileNumber	(Byte) Number of the opened file

This functions returns information about the File open mode

Return value	Open mode
1	INPUT
2	OUTPUT
8	APPEND
32	BINARY

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT, FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN, WRITE , INPUT , GETATTR

ASM

Calls	_FileAttr	
Input	r24: Filenumber	
Output	24: File open mode	r25: Errorcode
	C-Flag: Set on Error	

Partial Example
```
'open the file in BINARY mode
Open "test.biN" For Binary As #2
Print Fileattr(#2); " file mode"        ' should be 32 for binary
Put #2 , Sn                               ' write a single
Put #2 , Stxt                              ' write a string
Close #2
```

SEEK

Top Previous Next

Action

Function: Returns the position of the next Byte to be read or written
Statement: Sets the position of the next Byte to be read or written

Syntax

Function: NextReadWrite = **SEEK** (#bFileNumber)
Statement: **SEEk** #bFileNumber, NewPos

Remarks

bFileNumber	(Byte) Filenumber, which identifies an opened file
NextReadWrite	A Long Variable, which is assigned with the Position of the next Byte to be read or written (1-based)
NewPos	A Long variable that holds the new position the file pointer must be set too.

This function returns the position of the next Byte to be read or written. If an error occurs, 0 is returned. Check DOS-Error in variable gbDOSError.

The statement also returns an error in the gbDOSerror variable in the event that an error occurs. You can for example not set the file position behinds the file size.

In VB the file is filled with 0 bytes when you set the file pointer behind the size of the file. For embedded systems this does not seem a good idea.

Seek and Loc seems to do the same function, but take care : the seek function will return the position of the next read/write, while the Loc function returns the position of the last read/write. You may say that Seek = Loc+1.

 In QB/VB you can use seek to make the file bigger. When a file is 100 bytes long, setting the file pointer to 200 will increase the file with 0 bytes. By design this is not the case in AVR-DOS.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Function Calls	_FileSeek	
Input	r24: filenumber	X: Pointer to Long-variable, which gets the result
Output	r25: Errorcode	C-Flag: Set on Error

Statement Calls	_FileSeekSet	

Input	r24: filenumber	X: Pointer to Long-variable with the position
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example

```
Open "test.biN"for Binary As #2
Put#2 , B                                                    ' write a byte
Put#2 , W                                                    ' write a word
Put#2 , L                                                    ' write a long
Ltemp = Loc(#2) + 1                                          ' get the position of the
next byte
Print Ltemp ; " LOC"                                        ' store the location of
the file pointer
Print Seek(#2) ; " = LOC+1"
Close #2


'now open the file again and write only the single
Open "test.bin" For Binary As #2
Seek#2 , Ltemp                                              ' set the filepointer
Sn = 1.23                                                    ' change the single value
so we can check it better
Put #2 , Sn = 1                                              'specify the file
position
Close #2
```

BSAVE

Top Previous Next

Action
Save a range in SRAM to a File

Syntax
BSave sFileName, wSRAMPointer, wLength

Remarks

sFileName	(String) Name of the File to be written
wSRAMPointer	(Word) Variable, which holds the SRAM Address, from where SRAM should be written to a File
wLength	(Word) Count of Bytes from SRAM, which should be written to the file

This function writes a range from the SRAM to a file. A free file handle is needed for this function.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_BSave	
Input	X: Pointer to string with filename	Z: Pointer to Long-variable, which holds the start position of SRAM
	r20/r21: Count of bytes to be written	
Output	r25: Errorcode	C-Flag: Set on Error

Example
```
' THIS IS A CODE FRAGMENT, it needs AVR-DOS in order to work
'now the good old bsave and bload
Dim Ar(100)as Byte , I Asbyte
For I = 1 To 100
Ar(i) = I                                                  ' fill the array
Next

Wait 2

W = Varptr(ar(1))
Bsave"josef.img", W , 100
For I = 1 To 100
Ar(i) = 0                                                  ' reset the array
Next

Bload "josef.img" , W                                       ' Josef you are amazing !

For I = 1 To 10
Print Ar(i) ; " ";
```

```
Next
Print
```

BLOAD

Top Previous Next

Action
Writes the Content of a File into SRAM

Syntax
BLoad sFileName, wSRAMPointer

Remarks

sFileName	(String) Name of the File to be read
wSRAMPointer	(Word) Variable, which holds the SRAM Address to which the content of the file should be written

This function writes the content of a file to a desired space in SRAM. A free handle is needed for this function.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , KILL , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_BLoad	
Input	X: Pointer to string with filename	Z: Pointer to Long-variable, which holds the start position of SRAM
Output	r25: Errorcode	C-Flag: Set on Error

Example
```
' THIS IS A CODE FRAGMENT, it needs AVR-DOS in order to work
'now the good old bsave and bload
Dim Ar(100)as Byte , I Asbyte
For I = 1 To 100
Ar(i) = I                                                ' fill the array
Next

Wait 2

W = Varptr(ar(1))
Bsave"josef.img", W , 100
For I = 1 To 100
Ar(i) = 0                                                ' reset the array
Next

Bload "josef.img" , W                                     ' Josef you are amazing !

For I = 1 To 10
Print Ar(i) ; " ";
Next
Print
```

KILL

Top Previous Next

Action
Delete a file from the Disk

Syntax
KILL sFileName

Remarks

sFileName	A String variable or string expression, which denotes the file to delete

This function deletes a file from the disk. A file in use can't be deleted. WildCards in Filename are not supported. Check the DOS-Error in variable gDOSError.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , DISKFREE , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_DeleteFile	
Input	X: Pointer to string with filename	
Output	r25: Errorcode	C-Flag: Set on Error

Partial Example
```
'We can use the KILL statement to delete a file.
'A file mask is not supported
Print "Kill (delete) file demo"
Kill "test.txt"
```

DISKFREE

Top Previous Next

Action
Returns the free size of the Disk in KB.

Syntax
lFreeSize = **DISKFREE**()

Remarks

lFreeSize	A Long Variable, which is assigned with the available Bytes on the Disk in Kilo Bytes.

This functions returns the free size of the disk in KB.
With the support of FAT32, the return value was changed from byte into KB.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKSIZE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_GetDiskFreeSize
Input	none
Output	r16-r19: Long-Value of free Bytes

Partial Example
```
Dim Gbtemp1 As Byte                         ' scratch byte
Gbtemp1 =Initfilesystem(1)                  ' we must init the filesystem once
If Gbtemp1 > 0 Then
   Print#1 ,"Error "; Gbtemp1
Else
   Print#1 ," OK"
Print "Disksize : ";Disksize()        ' show disk size in bytes
Print "Disk free: ";Diskfree()         ' show free space too
End If
```

DISKSIZE

Top Previous Next

Action
Returns the size of the Disk in KB.

Syntax
lSize = **DISKSIZE**()

Remarks

lSize	A Long Variable, which is assigned with the capacity of the disk in Kilo Bytes

This functions returns the capacity of the disk in KB.
With the support of FAT32, the return value was changed from byte into KB.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

Calls	_GetDiskSize
Input	none
Output	16-r19: Long-Value of capacity in Bytes

Partial Example
```
Dim Gbtemp1 As Byte                         ' scratch byte
Gbtemp1 = Initfilesystem(1)                 ' we must init the filesystem once
If Gbtemp1 > 0 Then
   Print#1 ,"Error "; Gbtemp1
Else
   Print#1 ," OK"
Print "Disksize : "; Disksize()        ' show disk size in bytes
Print "Disk free: "; Diskfree()        ' show free space too
End If
```

GET

Top Previous Next

Action

Reads a byte from the hardware or software UART.
Reads data from a file opened in BINARY mode.

Syntax

GET #channel, var
GET #channel, var , [pos] [, length]

Remarks

GET in combination with the software/hardware UART reads one byte from the UART.
GET in combination with the AVR-DOS file system is very flexible and versatile. It works on files opened in BINARY mode and you can reads all data types.

#channel	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Var	The variable or variable array that will be assigned with the data from the file
Pos	This is an optional parameter that may be used to specify the position where the reading must start from. This must be a long variable.
Length	This is an optional parameter that may be used to specify how many bytes must be read from the file.

By default you only need to provide the variable name. When the variable is a byte, 1 byte will be read. When the variable is a word or integer, 2 bytes will be read. When the variable is a long or single, 4 bytes will be read. When the variable is a string, the number of bytes that will be read is equal to the dimensioned size of the string. DIM S as string * 10 , would read 10 bytes.

Note that when you specify the length for a string, the maximum length is 254. The maximum length for a non-string array is 65535.

Partial Example :

```
GET #1 , var ,,2          ' read 2 bytes, start at current position
GET #1, var , PS                ' start at position stored in long PS
GET #1, var , PS, 2         ' start at position stored in long PS and read 2 bytes
```

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT

ASM

current position	goto new position first
Byte:	
_FileGetRange_1 Input:	_FileGetRange_1 Input:

r24: File number X: Pointer to variable T-Flag cleared	r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Word/Integer:	
_FileGetRange_2 Input: r24: File number X: Pointer to variable T-Flag cleared	_FileGetRange_2 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Long/Single:	
_FileGetRange_4 Input: r24: File number X: Pointer to variable T-Flag cleared	_FileGetRange_4 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
String (<= 255 Bytes) with fixed length	
_FileGetRange_Bytes Input: r24: File number r20: Count of Bytes X: Pointer to variable T-Flag cleared	_FileGetRange_Bytes Input: r24: File number r20: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Array (> 255 Bytes) with fixed length	
_FileGetRange Input: r24: File number	_FileGetRange Input: r24: File number

r20/21: Count of Bytes X: Pointer to variable T-Flag cleared	r20/21: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set

Output from all kind of usage:
r25: Error Code
C-Flag on Error
X: requested info

Partial Example

```
'for the binary file demo we need some variables of different types
Dim B As Byte , W As Word , L As Long , Sn As Single , Ltemp As Long
Dim Stxt As String * 10
B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt = "test"

'open the file in BINARY mode
Open "test.biN"for Binary As #2
Put#2 , B ' write a byte
Put#2 , W ' write a word
Put#2 , L ' write a long
Ltemp = Loc(#2) + 1                                    ' get the position of the
next byte
Print Ltemp ; " LOC"                                   ' store the location of
the file pointer
Print Seek(#2) ; " = LOC+1"

Print Lof(#2) ; " length of file"
Print Fileattr(#2) ; " file mode"                      ' should be 32 for binary
Put #2 , Sn                                            ' write a single
Put #2 , Stxt                                          ' write a string

Flush #2                                               ' flush to disk
Close #2

'now open the file again and write only the single
Open "test.bin" For Binary As #2
L = 1 'specify the file position
B = Seek(#2 , L)                                       ' reset is the same as
using SEEK #2,L
Get#2 , B ' get the byte
Get#2 , W ' get the word
Get#2 , L ' get the long
Get#2 , Sn ' get the single
Get#2 , Stxt ' get the string
Close #2
```

PUT

Top Previous Next

Action

Writes a byte to the hardware or software UART.
Writes data to a file opened in BINARY mode.

Syntax

PUT #channel, var
PUT #channel, var ,[pos] [,length]

Remarks

PUT in combination with the software/hardware UART is provided for compatibility with BASCOM-8051. It writes one byte

PUT in combination with the AVR-DOS file system is very flexible and versatile. It works on files opened in BINARY mode and you can write all data types.

#channel	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Var	The variable or variable array that will be written to the file
Pos	This is an optional parameter that may be used to specify the position where the data must be written. This must be a long variable.
Length	This is an optional parameter that may be used to specify how many bytes must be written to the file.

By default you only need to provide the variable name. When the variable is a byte, 1 byte will be written. When the variable is a word or integer, 2 bytes will be written. When the variable is a long or single, 4 bytes will be written. When the variable is a string, the number of bytes that will be written is equal to the dimensioned size of the string. DIM S as string * 10 , would write 10 bytes.

Note that when you specify the length for a string, the maximum length is 255. The maximum length for a non-string array is 65535.

Example

```
PUT #1, var
PUT #1, var , , 2  ' write 2 bytes at default position
PUT #1, var ,PS, 2  ' write 2 bytes at location storied in variable PS
```

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET, FILEDATE , FILETIME , FILEDATETIME , DIR , FILELEN , WRITE , INPUT, AVR-DOS File system

ASM

current position	Goto new position first
Byte:	

_FilePutRange_1 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_1 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Word/Integer:	
_FilePutRange_2 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_2 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Long/Single:	
_FilePutRange_4 Input: r24: File number X: Pointer to variable T-Flag cleared	_FilePutRange_4 Input: r24: File number X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
String (<= 255 Bytes) with fixed length	
_FilePutRange_Bytes Input: r24: File number r20: Count of Bytes X: Pointer to variable T-Flag cleared	_FilePutRange_Bytes Input: r24: File number r20: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set
Array (> 255 Bytes) with fixed length	
_FilePutRange Input: r24: File number r20/21: Count of Bytes X: Pointer to variable T-Flag cleared	_FilePutRange Input: r24: File number r20/21: Count of bytes X: Pointer to variable r16-19 (A): New position (1-based) T-Flag Set

Output from all kind of usage:
r25: Error Code
C-Flag on Error

Example

```
'for the binary file demo we need some variables of different types
Dim B AsByte, W AsWord, L AsLong, Sn AsSingle, Ltemp AsLong
```

```
Dim Stxt AsString* 10
B = 1 : W = 50000 : L = 12345678 : Sn = 123.45 : Stxt ="test"

'open the file in BINARY mode
Open "test.biN" For Binary As#2
Put#2 , B                ' write a byte
Put#2 , W                ' write a word
Put#2 , L                ' write a long
Ltemp =Loc(#2)+ 1 ' get the position of the next byte
Print Ltemp ;" LOC"' store the location of the file pointer
Print Seek(#2);" = LOC+1"

PrintLof(#2);" length of file"
PrintFileattr(#2);" file mode"' should be 32 for binary
Put#2 , Sn                ' write a single
Put#2 , Stxt        ' write a string

Flush#2                ' flush to disk
Close#2

'now open the file again and write only the single
Open "test.bin" For Binary As #2
L = 1 'specify the file position
B =Seek(#2 , L)        ' reset is the same as using SEEK #2,L
Get#2 , B                ' get the byte
Get#2 , W                ' get the word
Get#2 , L                ' get the long
Get#2 , Sn                ' get the single
Get#2 , Stxt        ' get the string
Close#2
```

FILEDATE

Top Previous Next

Action
Returns the date of a file

Syntax
sDate = **FILEDATE** ()
sDate = **FILEDATE** (file)

Remarks

Sdate	A string variable that is assigned with the date.
File	The name of the file to get the date of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE, GET , PUT, FILELEN , FILETIME , FILEDATETIME , DIR , WRITE , INPUT

ASM

Calls	_FileDateS ; with filename	_FileDateS0 ; for current file from DIR()
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Partial Example
```
Print "File demo"
Print Filelen("josef.img");" length"           ' length of file
Print Filetime("josef.img");" time"            ' time file was changed
Print Filedate("josef.img");" date"           ' file date
```

FILELEN

Top Previous Next

Action
Returns the size of a file

Syntax
lSize = **FILELEN** ()
lSize = **FILELEN** (file)

Remarks

lSize	A Long Variable, which is assigned with the file size in bytes of the file.
File	A string or string constant to get the file length of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , WRITE , INPUT

ASM

Calls	_FileLen
Input	
Output	

Partial Example
```
Print "File demo"
Print Filelen("josef.img");" length"          ' length of file
Print Filetime("josef.img");" time"          ' time file was changed
Print Filedate("josef.img");" date"          ' file date
```

FILETIME

Top Previous Next

Action
Returns the time of a file

Syntax
sTime = **FILETIME** ()
sTime = **FILETIME** (file)

Remarks

Stime	A string variable that is assigned with the file time.
File	The name of the file to get the time of.

This function works on any file when you specify the filename. When you do not specify the filename, it works on the current selected file of the DIR() function.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT , FILELEN , FILEDATE , FILEDATETIME , DIR , WRITE , INPUT

ASM

Calls	_FileTimeS ; with file param	_FileTimeS0 ; current file
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Example
```
Print "File demo"
Print Filelen("josef.img");" length"         ' length of file
Print Filetime("josef.img");" time"         ' time file was changed
Print Filedate("josef.img");" date"         ' file date
```

FILEDATETIME

Top Previous Next

Action
Returns the file date and time of a file

Syntax
Var = **FILEDATETIME** ()
Var = **FILEDATETIME** (file)

Remarks

Var	A string variable or byte array that is assigned with the file date and time of the specified file
File	The name of the file to get the date time of.

When the target variable is a string, it must be dimensioned with a length of at least 17 bytes.
When the target variable is a byte array, the array size must be at least 6 bytes.

When you use a numeric variable, the internal file date and time format will be used.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT , FILELEN , FILEDATE , FILETIME , DIR , WRITE , INPUT

ASM

Calls	_FileDateTimeS	_FileDateTimeS0
Input		
Output		

Calls	_FileDateTimeB	_FileDateTimeB0
Input		
Output		

Example
See fs_subfunc_decl_lib.bas in the samples dir.

DIR

Top Previous Next

Action

Returns the filename that matches the specified file mask.

Syntax

sFile = **DIR**(mask)
sFile = **DIR**()

Remarks

SFile	A string variable that is assigned with the filename.
Mask	A file mask with a valid DOS file mask like *.TXT Use *.* to select all files.

The first function call needs a file mask. All other calls do not need the file mask. In fact when you want to get the next filename from the directory, you must not provide a mask after the first call.

Dir() returns an empty string when there are no more files or when no file name is found that matches the mask.

See also

INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , DISKSIZE , GET , PUT, FILELEN , FILEDATE , FILETIME , FILEDATETIME , WRITE , INPUT , MKDIR, RMDIR , CHDIR

ASM

Calls	_Dir ; with file mask	_Dir0 ; without file mask
Input	X : points to the string with the mask	Z : points to the target variable
Output		

Partial Example

```
'Lets have a look at the file we created
Print "Dir function demo"
S = Dir("*.*")
'The first call to the DIR() function must contain a file mask
' The * means everything.
'
While Len(s)> 0 ' if there was a file found
  Print S ;" ";Filedate();" ";Filetime();" ";Filelen()
' print file , the date the fime was created/changed , the time and the size of the file
  S = Dir()' get next
Wend
```

WRITE

Top Previous Next

Action
Writes data to a sequential file

Syntax
WRITE #ch , data [,data1]

Remarks

Ch	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.
Data , data1	A variable who's content are written to the file.

When you write a variables value, you do not write the binary representation but the ASCII representation. When you look in a file it contains readable text.

When you use PUT, to write binary info, the files are not readable or contain unreadable characters. Strings written are surrounded by string delimiters "". Multiple variables written are separated by a comma. Consider this example :

```
Dim S as String * 10 , W as Word
S="hello" : W = 100
OPEN "test.txt" For OUTPUT as #1
WRITE #1, S , W
CLOSE #1
```

The file content will look like this : "hello",100
Use INPUT to read the values from value.

See also
INITFILESYSTEM , OPEN , CLOSE, FLUSH , PRINT, LINE INPUT, LOC, LOF , EOF , FREEFILE , FILEATTR , SEEK , BSAVE , BLOAD , KILL , DISKFREE , GET , PUT , FILEDATE , FILETIME , FILEDATETIME , DIR , WRITE , INPUT

ASM

Calls	_FileWriteQuotationMark	_FileWriteDecInt
	_FileWriteDecByte	_FileWriteDecWord
	_FileWriteDecLong	_FileWriteDecSingle
Input	Z points to variable	
Output		

Partial Example

```
Dim S As String * 10 , W As Word , L As Long

S = "write"
Open "write.dmo"for Output As #2
Write #2 , S , W , L                                    ' write is also supported
Close #2
```

```
Open "write.dmo"for Input As #2
Input #2 , S , W , L                                    ' write is also supported
Close #2
Print S ; " " ; W ; " " ; L
```

INPUT

Top Previous Next

Action
Allows input from the keyboard, file or SPI during program execution.

Syntax
INPUT [" prompt"] , var[, varn]
INPUT #ch, var[, varn]

Remarks

Prompt	An optional string constant printed before the prompt character.
Var,varn	A variable to accept the input value or a string.
Ch	A channel number, which identifies an opened file. This can be a hard coded constant or a variable.

The INPUT routine can be used when you have an RS-232 interface on your uP.
The RS-232 interface can be connected to a serial communication port of your computer.
This way you can use a terminal emulator and the keyboard as an input device.
You can also use the built-in terminal emulator.

For usage with the AVR-DOS file system, you can read variables from an opened file. Since these variables are stored in ASCII format, the data is converted to the proper format automatically.
When you use INPUT with a file, the prompt is not supported.

When $BIGSTRINGS is used you can read read up to 65535 bytes.

Difference with VB
In VB you can specify **&H** with INPUT so VB will recognize that a hexadecimal string is being used.
BASCOM implements a new statement : INPUTHEX.

Xmega-SPI
When receiving data from the SPI interface, you need to activate the SS pin. Some chips might need an active low, others might need an active high. This will depends on the slave chip.
When you use the SS=AUTO option, the level of SS will be changed automatic. Thus SS is made low, then the data bytes are received, and finally , SS is made high again.

Receiving data works by sending a data byte and returning the data that is shifted out. The data that will be sent is a 0. You can alter this in the library, _inputspivar routine.
You can not sent constants using the INPUT with SPI. So INPUT #10, "SPI", var is not supported.
INPUT used with SPI will not wait for a return either. It will wait for the number of bytes that fits into the variable.
See CONFIG SPIx for an example.

Number of Bytes
The compiler will receive 1 byte for a variable which was dimensioned as a BYTE.
It will receive 2 bytes for a WORD/INTEGER, 4 bytes for a LONG/SINGLE and 8 bytes for a DOUBLE.
As with all routines in BASCOM, the least significant Byte will be received first.

If you specify an array, it will receive one element.
With an optional parameter you can provide how many bytes must be received. You must use a semicolon (;) to specify this parameter. This because the comma (,) is used to receive multiple variables.

Sample
```
Dim Tmparray(5) As Byte  , Spi_send_byte As Byte , W as Word
Input #12 , Spi_receive_byte ; 1                    ' READ 1 byte
Input #12 , Tmparray(1) ; 1 , Tmparray(2) ; B   ' read 1 byte and 'b' bytes starting at element 2
```

See also

INPUTHEX , PRINT , ECHO , WRITE , INPUTBIN

Example

```
'-----------------------------------------------------------------------------
'name                     : input.bas
'copyright                : (c) 1995-2005, MCS Electronics
'purpose                  : demo: INPUT, INPUTHEX
'micro                    : Mega48
'suited for demo          : yes
'commercial addon needed  : no
'-----------------------------------------------------------------------------

$regfile = "m48def.dat"                                    ' specify the used micro
$crystal = 4000000                                          ' used crystal frequency
$baud = 19200                                               ' use baud rate
$hwstack = 32                                               ' default use 32 for the hardware
stack
$swstack = 10                                               ' default use 10 for the SW stack
$framesize = 40                                             ' default use 40 for the frame space

Dim V As Byte , B1 As Byte
Dim C As Integer , D As Byte
Dim S As String * 15

Input "Use this to ask a question " , V
Input B1                                                    'leave out for no question

Input "Enter integer " , C
Print C


Inputthex "Enter hex number (4 bytes) " , C
Print C
Inputthex "Enter hex byte (2 bytes) " , D
Print D

Input "More variables " , C , D
Print C ; " " ; D

Input C Noecho                                              'supress echo

Input "Enter your name " , S
Print "Hello " ; S

Input S Noecho                                              'without echo
Print S
End
```