

Table of Contents

- [xBIOS Overview](#)
- [General Notes](#)
- [Download Location](#)
- [Autorun](#)
- [Filename Format](#)
- [xBIOS Limitations](#)
- [Default Memory Layout](#)
- [Disk Images](#)
- [Practical Usage](#)
- [Examples](#)
- [xBIOS_RENAME_ENTRY](#)
- [xBIOS_LOAD_FILE](#)
- [xBIOS_OPEN_FILE](#)
- [xBIOS_LOAD_DATA](#)
- [xBIOS_WRITE_DATA](#)
- [xBIOS_OPEN_CURRENT_DIR](#)
- [xBIOS_GET_BYTE](#)
- [xBIOS_PUT_BYTE](#)
- [xBIOS_FLUSH_BUFFER](#)
- [xBIOS_SET_LENGTH](#)
- [xBIOS_SET_INIAD](#)
- [xBIOS_SET_FILE_OFFSET](#)
- [xBIOS_SET_RUNAD](#)
- [xBIOS_SET_DEFAULT_DEVICE](#)
- [xBIOS_OPEN_DIR](#)
- [xBIOS_LOAD_BINARY_FILE](#)
- [xBIOS_OPEN_DEFAULT_DIR](#)
- [xBIOS_SET_DEVICE](#)
- [xBIOS_RELOCATE_BUFFER](#)
- [xBIOS_GET_ENTRY](#)
- [xBIOS_OPEN_DEFAULT_FILE](#)
- [xBIOS_READ_SECTOR](#)
- [xBIOS_FIND_ENTRY](#)
- [xBIOS_SET_BUFFER_SIZE](#)
- [Counters During I/O](#)
- [Playing Music During I/O](#)
- [Indexed Data within File - LZ4 Graphics Decompression](#)
- [Use of High Speed Devices](#)
- [Detecting High Speed Devices](#)
- [Using Alternative Disk Drives](#)
- [Checking if Alternative Disk Drives Exist](#)
- [Varying Directory Sizes](#)
- [xBOOT](#)

xBIOS Overview#

xBIOS is like a programmers version of DOS. With it you can easily access files from your programs without using Atari DOS. It is smaller than DOS and therefore saves memory in your programs. You can even run programs from as low as \$0200, however \$0800 or \$2000 are more common.

General Notes#

In this section you will find some general notes about xBIOS and how it functions.

Download Location#

<http://xxl.atari.pl>

Autorun#

To allow an executable to auto run on bootup, have it named as ?XAUTORUN?. If no such file exists on the disk, the xBIOS loader menu will appear.

Filename Format#

The filenames are in 8.3 format but without the dot. Filenames are padded out to 11 characters and are converted to uppercase. Therefore, ?file.txt? will become:

```
?FILE      TXT?  (4 letters, 4 spaces and a 3 character extension)
```

xBIOS is case insensitive.

xBIOS Limitations#

- a) New files cannot be created.
- b) Existing files cannot be extended in length.
- c) New directories cannot be created.

Default Memory Layout#

\$0000-\$00FF	Free
\$0200-\$06FF	Free
\$0700-\$07FF	xBIOS I/O buffer
\$0800-\$0BFF	xBIOS
\$0C00-\$CFFF	Free
\$D000-\$D7FF	Atari H/W
\$D800-\$FFFF	Free

Disk Images#

Disk images (.atr files) need to be prepared in advance for xBIOS to work.

A disk image is a file which simulates a disk. It will contain 1 or more files which can be used by your programs.

In order to prepare them, an external tool needs to be used.

?dir2atr? : <http://www.horus.com/~hias/atari>

A tool for creating an ATR file from a folder.

?franny? : <http://atariage.com/forums/topic/159325-program-to-add-to-and-extract-files-from-atr/>

Allows the additional of individual files to an ATR file.

Practical Usage#

In order to have a disk which is usable, a project build system should complete the following steps:

- a) Make a copy of the xBIOS.atr file, renaming the new file to something relevant for your project.
- b) Add all the new files to the file. ?franny? and ?dir2atr? are good tools for this. Ensure that the program which you want to initially run is named ?XAUTORUN?.

Example Windows batch file:

```
mads myProg.asm
copy /Y myProg.obx myatr\XAUTORUN
dir2atr -md -B xboot.obx myatr.atr myatr
pause
```

xBIOS now also needs adding into the development project which is being created.

There are two ways of achieving this:

Option 1: The following code needs to be pasted into a project:

(Valid for xBIOS v4.3)

```
xBIOS equ $800
xBIOS_VERSION equ xBIOS+$02
xBIOS_RENAME_ENTRY equ xBIOS+$03
xBIOS_LOAD_FILE equ xBIOS+$06
xBIOS_OPEN_FILE equ xBIOS+$09
xBIOS_LOAD_DATA equ xBIOS+$0c
xBIOS_WRITE_DATA equ xBIOS+$0f
xBIOS_OPEN_CURRENT_DIR equ xBIOS+$12
xBIOS_GET_BYTE equ xBIOS+$15
xBIOS_PUT_BYTE equ xBIOS+$18
xBIOS_FLUSH_BUFFER equ xBIOS+$1b
xBIOS_SET_LENGTH equ xBIOS+$1e
xBIOS_SET_INIAD equ xBIOS+$21
xBIOS_SET_FILE_OFFSET equ xBIOS+$24
xBIOS_SET_RUNAD equ xBIOS+$27
xBIOS_SET_DEFAULT_DEVICE equ xBIOS+$2a
xBIOS_OPEN_DIR equ xBIOS+$2d
xBIOS_LOAD_BINARY_FILE equ xBIOS+$30
xBIOS_OPEN_DEFAULT_DIR equ xBIOS+$33
xBIOS_SET_DEVICE equ xBIOS+$36
xBIOS_RELOCATE_BUFFER equ xBIOS+$39
xBIOS_GET_ENTRY equ xBIOS+$3c
xBIOS_OPEN_DEFAULT_FILE equ xBIOS+$3f
xBIOS_READ_SECTOR equ xBIOS+$42
xBIOS_FIND_ENTRY equ xBIOS+$45
xBIOS_SET_BUFFER_SIZE equ xBIOS+$48

xDIRSIZE equ xBIOS+$3e5 ; current directory size in sectors (1 byte)
xSPEED equ xBIOS+$3e6 ; STANDARD SPEED (1 byte)
xHSPEED equ xBIOS+$3e7 ; ULTRA SPEED (1 byte)
xIRQEN equ xBIOS+$3e8 ; User IRQ (1 byte)
xAUDCTL equ xBIOS+$3e9 ; AUDCTL
xFILE equ xBIOS+$3ea ; File handle (2 bytes)
xDIR equ xBIOS+$3ec ; Root directory handle (2 bytes)
xIOV equ xBIOS+$3ee ; I/O module entry (2 bytes)
```

```

xBUFFERH      equ xBIOS+$3f0 ; Buffer adr hi byte (1 byte)
xBUFSIZE      equ xBIOS+$3f1 ; Buffer size lo byte $100-SIZE (1 byte)
xDAUX3        equ xBIOS+$3f2 ; Buffer offset (1 byte)
xSEGMENT      equ xBIOS+$3f3 ; Bytes to go in binary file segment (2 bytes)
xNOTE         equ xBIOS+$3f5 ; File pointer (3 bytes)
xDEVICE       equ xBIOS+$3fc ; Device ID
xDCMD         equ xBIOS+$3fd ; CMD (1 byte)
xDAUX1        equ xBIOS+$3fe ; Sector lo byte (1 byte)
xDAUX2        equ xBIOS+$3ff ; Sector hi byte (1 byte)

```

Each entry in this jump table specifies where in memory that the code for each function resides. If you ever wish to start from a different address, change the value for ?xBIOS? above and change the value in the binary header of xBIOS.com.

Option 2: From xBIOS v4.2 and above, you can use the xBIOS.cfg file which is as below.

```

opt h-
.byte   $43           ; config for v4.3 version
.byte   c'XAUTORUN   ' ; autorun file fname
.byte   >$0800       ; xB adress for relocater
.byte   >$0700       ; xB buffer adress for relocater
.word   $02e2        ; INITAD
.word   $02e0        ; RUNAD
.word   $0000        ; custom I/O module - $0000 means use internal xB SIO
.word   $0000        ; relocater for custom I/O module
.byte   $ff          ; PORTB - you can't swith off BASIC this way
.byte   $40          ; NMIEN - config at start
.byte   $c0          ; IRQEN - config at start

```

Please note that xBIOS do not use page zero. You can change the start address from the third entry above. Now that the above exists, the following examples can be used to access files in the ATR file.

Examples#

Here are some of the examples taken from <http://xxl.atari.pl> and other online sources, but with a few extra/modified comments in English.

xBIOS_RENAME_ENTRY#

This function allows you to rename a file or directory. There is no limit to the characters used in the filename apart from that they must fit a case insensitive ?8.3? format without the dot. If your filename is not 8 characters long, pad it out with spaces.

```

ldy <fname
ldx >fname
JSR xBIOS_RENAME_ENTRY
fname
.byte c'FILENAME_SRC ' ; 11 char ATASCII (8.3 without the dot, space padded)
.byte c'FILENAME_DST ' ; 11 char ATASCII (8.3 without the dot, space padded)

```

xBIOS_LOAD_FILE#

Load and run the file, INIT and RUN headers are supported. "Boot Loader" has an analogous function xBOOT_LOAD_FILE. In the case of the boot loader if the file does not have a defined block RUN will be launched from the beginning of the first block.

```
ldy <fname
ldx >fname
JSR xBIOS_LOAD_FILE
fname .byte c'MYFILE COM'; 11 characters ATASCII (8.3 without the dot, space padded)
```

xBIOS_OPEN_FILE#

Open a file in order to carry out subsequent IO operations.

```
ldy <fname
ldx >fname
JSR xBIOS_OPEN_FILE
fname .byte c'MYFILE COM'; 11 characters ATASCII (8.3 without the dot, space padded)
```

xBIOS_LOAD_DATA#

Load data from file to a specified address. You can set the file offset (FILE_OFFSET) and the amount of data to be loaded (SET_LENGTH). If you do not define these values, data will be loaded from the current position of the file pointer to the end of the file. If you want to load a binary file using the headers defined in it or use the functions xBIOS_BINARY_LOAD or xBIOS_LOAD_FILE.

```
ldy <dest
ldx >dest
JSR xBIOS_LOAD_DATA
```

xBIOS_WRITE_DATA#

Save data from memory to a file, starting from the current position in the file. You can set the file pointer offset current (FILE_OFFSET) and the amount of data to be saved (SET_LENGTH). If you do not define these values, data from the current file position to the end of the file is written to the file.

```
ldy <src
ldx >src
JSR xBIOS_WRITE_DATA
```

xBIOS_OPEN_CURRENT_DIR#

Opens the current directory.

```
JSR xBIOS_OPEN_CURRENT_DIR
```

xBIOS_GET_BYTE#

The next byte of an open file is loaded into the register A (accumulator).

```
JSR xBIOS_GET_BYTE
```

xBIOS_PUT_BYTE#

The byte value in the accumulator is written to a file at its current pointer location.

```
lda BYTE
JSR xBIOS_PUT_BYTE
```

xBIOS_FLUSH_BUFFER#

All write operations are cached, use this to flush the buffer to the current file.

```
JSR xBIOS_FLUSH_BUFFER
```

xBIOS_SET_LENGTH#

Defines the amount of data to process while reading or writing. If your OPEN_FILE value is not defined this operation will be carried out until the end of the file.

```
ldy <len  
ldx >len  
JSR xBIOS_SET_LENGTH
```

xBIOS_SET_INIAD#

Allows you to change the init address vector INITAD (\$2E2) for loaded binary files.

```
ldy <addr  
ldx >addr  
JSR xBIOS_SET_INIAD
```

xBIOS_SET_FILE_OFFSET#

Sets the current read/write position in the current file with a value stored in A, X, Y. This item is calculated relative to the beginning of the file. In DOS speak, the operation is called "POINT".

```
ldy <pos  
ldx >pos  
lda ^pos  
JSR xBIOS_SET_FILE_OFFSET
```

xBIOS_SET_RUNAD#

Allows you to change the run address vector RUNAD (\$2E0) for loaded binary files.

```
ldy <addr  
ldx >addr  
JSR xBIOS_SET_RUNAD
```

xBIOS_SET_DEFAULT_DEVICE#

Restores the standard IO device.

```
JSR xBIOS_SET_DEFAULT_DEVICE
```

xBIOS_OPEN_DIR#

Allows you to change the current directory.

```
ldy <fname  
ldx >fname  
JSR xBIOS_OPEN_DIR
```

Note that the directory cannot be created by xBIOS and therefore must be created with an external tool such as ?franny?.

xBIOS_LOAD_BINARY_FILE#

Load and run the binary file from the current read/write position. INIT and RUN headers are supported.

```
JSR xBIOS_LOAD_BINARY_FILE
```

xBIOS_OPEN_DEFAULT_DIR#

Opens the default directory.

```
JSR xBIOS_OPEN_DEFAULT_DIR
```

xBIOS_SET_DEVICE#

Change the IO device.

```
ldy <dev  
ldx >dev  
JSR xBIOS_SET_DEVICE
```

xBIOS_RELOCATE_BUFFER#

Change address IO buffer. If before the call to set the marker C = 1, the relocation can be carried out even during IO. The data will not be lost. If the marker before calling C = 0, buffer contents will not be copied to a new location.

```
ldx >buffer  
JSR xBIOS_RELOCATE_BUFFER
```

xBIOS_GET_ENTRY#

Gets another entry in the directory. The X register returns the index to the filename or folder (byte of buffer address is stored in the variable xBUFFERH). The accumulator is set with the status. The carry flag is set when the end of the directory is found.

```
JSR xBIOS_GET_ENTRY
```

xBIOS_OPEN_DEFAULT_FILE#

Opens the default file. The function does not search the directory, the file handle is derived from the variable 'Xfile'.

```
JSR xBIOS_OPEN_DEFAULT_FILE
```

xBIOS_READ_SECTOR#

Load a sector into a buffer. ldx >sector ; High byte of the sector number ldy <sector ; Low byte of the sector number

```
JSR xBIOS_READ_SECTOR
```

xBIOS_FIND_ENTRY#

This function allows you to find the specified directory entry. The X register returns the index to the filename or folder (byte of buffer address is stored in the variable xBUFFERH). The accumulator is the status byte. If an entry is not found, the carry flag is set.

```
ldy <fname  
ldx >fname  
JSR xBIOS_FIND_ENTRY
```

xBIOS_SET_BUFFER_SIZE#

This feature allows you to set the buffer size for IO operations. Buffer Size is also stored in the variable xBUFSIZE in bytes format. Ida # \$ 100-SIZE

```
JSR xBIOS_SET_BUFFER_SIZE
```

Counters During I/O#

```
xIRQEN      equ xBIOS+$3ea ; User IRQ (1 byte)
xSEGMENT    equ xBIOS+$3f4 ; Bytes to go in binary file segment (2 bytes)
xNOTE       equ xBIOS+$3f6 ; File pointer (3 bytes)

PUPBT1      equ $033D      ; Power-up validation byte 1

            icl      'atarihw.ah'

            opt      h+o+l+

start_sio    org      $c00
            sei
            lda      #$00
            sta      nmien
            sta      irqen
            sta      xIRQEN
            sta      dmactl
            lda      #$fe
            sta      portb
            lda      <MyNMI
            sta      nmiv
            lda      >MyNMI
            sta      nmiv+1
            lda      <mydlist
            sta      dlistl
            lda      >mydlist
            sta      dlisth
            jsr      xBIOS_SET_DEFAULT_DEVICE ; SIO drive
            ldx      >RESETV
            ldy      <RESETV
            jmp      xBIOS_SET_RUNAD      ; new runad vector
            ini      start_sio

            org      RESETV
            .word    run_adr              ; RUN

            org      PUPBT1
            .byte    d'xxl'              ; Let the RESET key work as a RESET key

            org      $0000

run_adr      lda      #$74
            sta      colbak

_stop       jmp      _stop              ; Endless loop

mydlist      :12 .byte $70
            .byte $46,<counter,>counter
            .byte $06
            .byte $41,<mydlist,>mydlist
```



```

counter      .byte d'SEGMENT:      '
             .byte d'FILE OFFSET:  '

start        sei
             ldx      #3
             ldy      #0
@            inc      PORTB
             lda      $e000,y
_csrc        equ      *-1
             dec      PORTB
             sta      (_csrc-1),y
_cdst        equ      *-1
             dey
             bne      @-
             inc      _csrc
             dex
             bpl      @-
@            lda      vcount
             bne      @-
             lda      #$22
             sta      dmactl
             lda      #$40
             sta      nmien
             rts

MyNMI        sta      nmiA
             sty      nmiY
             inc      licz
             lda      #0
licz         equ      *-1
             lsr      @
             bcc      secondnmi

firstnmi     ldy      #16
             lda      xSEGMENT+1
             jsr      puthex
             lda      xSEGMENT
             jsr      puthex
             jmp      endnmi

secondnmi    ldy      #20+14
             lda      xNOTE+2
             jsr      puthex
             lda      xNOTE+1
             jsr      puthex
             lda      xNOTE
             jsr      puthex

endnmi       lda      #0
nmiA         equ      *-1
             ldy      #0
nmiY         equ      *-1
             rti

puthex       pha
             lsr      @
             lsr      @
             lsr      @
             lsr      @

```

```

nibble      jsr      nibble
             pla
             and      #$0f
             cmp      #$0a
             sed
             adc      #$10
             cld
             sta      counter,y
             iny
             rts

```

```
ini start
```

```
org $0c00
```

```
:$c300      .byte $ff
```

Playing Music During I/O#

```
opt h+o+l+
```

```
org      PUPBT1
.byte    d'xxl'
```

```
; Let the RESET key work as a RES
```

```

start_sio   org      $c00
             sei
             lda      #$00
             sta      nmien
             sta      irqen
             sta      xIRQEN
             sta      dmactl
             lda      #$fe
             sta      portb
             lda      <MyNMI
             sta      nmiv
             lda      >MyNMI
             sta      nmiv+1
             lda      #$40
             sta      nmien
             jmp      xBIOS_SET_DEFAULT_DEVICE ; SIO drive

```

```

MyNMI      sta      vbisaveA+1
             stx      vbisaveX+1

```

```

speed      lda      #0
             bne      wp

```

```

loadpat    ldx      #0
             lda      pat,x
             bmi      pass
             asl      @
             asl      @
             asl      @
             sta      wp+1

```

```

pass       inx
             lda      #$3f
             sax      loadpat+1
wp          ldx      #0

```

```

        lda     ins,x
        sta     audc1
        lda     frq,x
        sta     audf1
        inc     wp+1
        dec     speed+1
        bpl     pass2
        lda     #5
        sta     speed+1
wp1     lda     #$20
        sta     pass2+1
        eor     #$60
        sta     wp1+1
pass2   ldx     #0
        lda     ins,x
        sta     audc2
        inc     pass2+1

vbisaveA   lda     #0
vbisaveX   ldx     #0
           rti

ins        .byte   $0F,$AF,$AC,$A7,$A2,$00,$04,$A3
           .byte   $22,$21,$A1,$00,$00,$00,$00,$00
           .byte   $AF,$AF,$09,$07,$05,$04,$04,$03
           .byte   $03,$03,$02,$02,$02,$01,$01,$00
           .byte   $87,$84,$83,$82,$81,$00,$00,$00
           .byte   $00,$00,$00,$00,$00,$00,$00,$00
           .byte   $84,$84,$84,$84,$84,$84,$83,$83
           .byte   $83,$83,$82,$82,$81,$81,$00,$00
           .byte   $83,$82,$81,$81,$81,$00,$00,$00
           .byte   $00,$00,$00,$00,$00,$00,$00,$00

frq        .byte   $04,$C0,$D0,$E0,$F0,$00,$04,$C0
           .byte   $D0,$E0,$F0,$00,$00,$00,$00,$00
           .byte   $98,$A8,$03,$03,$03,$03,$03,$03
           .byte   $03,$03,$03,$03,$03,$03,$03,$03
           .byte   $00,$00,$00,$00,$00,$00,$00,$00
           .byte   $00,$00,$00,$00,$00,$00,$00,$00
           .byte   $00,$00,$00,$00,$00,$00,$00,$00
           .byte   $00,$00,$00,$00,$00,$00,$00,$00

pat        .byte   $00,$80,$00,$80,$01,$80,$00,$00
           .byte   $80,$00,$00,$80,$01,$80,$03,$80
           .byte   $00,$80,$00,$80,$01,$80,$02,$02
           .byte   $00,$80,$01,$01,$80,$00,$03,$80
           .byte   $00,$80,$00,$80,$01,$80,$00,$00
           .byte   $80,$00,$00,$80,$01,$80,$03,$80
           .byte   $00,$00,$00,$80,$01,$00,$00,$80
           .byte   $00,$80,$01,$01,$80,$00,$01,$00

ini        start_sio

           org     $1000
:$C000    .byte   $ff
           org     $D800
:$2700    .byte   $ff

run_adr    lda     #$74

```

```

        sta      colbak
_stop   jmp      _stop      ; endless loop

        run      run_adr

```

Indexed Data within File - LZ4 Graphics Decompression#

```

xFILE      equ      xBIOS+$3ec ; File handle
xDAUX3     equ      xBIOS+$3f3 ; Buffer offset if AtariDOS FS
xDAUX2     equ      xBIOS+$3fd ; Sector hi if AtariDOS FS
xDAUX1     equ      xBIOS+$3fe ; Sector lo if AtariDOS FS
PUPBT1     equ      $033D      ; Power-up validation byte 1
myscr      equ      $8150
myscr1     equ      $9000
colour     equ      myscr+$1e01
fhandle    equ      $80

        org      PUPBT1
        .byte   d'xxl'      ; Let the RESET key work as a RESET key

        org      $c00

Mydlist    .byte   $70,$70,$70
           .byte   $4e,a(myscr)
:93        .byte   $0e
           .byte   $4e,a(myscr1)
:97        .byte   $0e
           .byte   $41,a(Mydlist)

myinit     lda      #$00
           sta      DMACTL5
           lda      <Mydlist
           sta      DLISTLS
           lda      >Mydlist
           sta      DLISTHS
           lda      #%00100010
           sta      DMACTL5

load_index  ldx      load_gfx+1
           lda      xDAUX1
           sta      fhandle,x
           lda      xDAUX2
           sta      fhandle+1,x
           lda      xDAUX3
           sta      fhandle+2,x
           cpx      #$0c
           bne      skip
           lda      #$100-(ldgfx+2-load_gfx)
           sta      ldgfx+1
           bne      skip

load_gfx   ldx      #0
           lda      fhandle,x
           sta      xFILE
           lda      fhandle+1,x
           sta      xFILE+1
           jsr      xBIOS_OPEN_DEFAULT_FILE
           ldx      load_gfx+1

```

```

        lda     fhandle+2,x
        sta     xDAUX3
skip    lda     #%1111
        sbx     #$100-$04
        sax     load_gfx+1
        lda     <myscr
        sta     dest
        lda     >myscr
        sta     dest+1
        jsr     unlz4
        ldx     #$02
@       lda     colour,x
        sta     COLPF0S,x
        dex
        bpl     @-
ldgfx   bmi     load_index

unlz4   icl     'unlz4.asm'

        ini     myinit

        opt     h-
        ins     'DRUID.LZ4', $0b, .FILESIZE 'DRUID.LZ4'-$0b-$06
        ins     'LOADING.LZ4', $0b, .FILESIZE 'LOADING.LZ4'-$0b-$06
        ins     'SHPOON2.LZ4', $0b, .FILESIZE 'SHPOON2.LZ4'-$0b-$06
        ins     'POTATERR.LZ4', $0b, .FILESIZE 'POTATERR.LZ4'-$0b-$06

```

Use of High Speed Devices#

When you first load xBIOS, you can complete a check to see if your disk drive is ultraspeed (see next example). If so, save a value to xHSPEED. When loading data, copy the value of xHSPEED to xSPEED and use the following code:

```

lda xHSPEED
bmi no_hispeed
sta xSPEED

```

This works in conjunction with: "jsr xBIOS_SET_DEFAULT_DEVICE" If you are using the AtariOS I/O module, speed control is not possible.

Detecting High Speed Devices#

If you want to detect if the user's disk drive has high speed capabilities, you can detect it's high speed status using this code.

```

xBIOS_SET_DEFAULT_DEVICE equ xBIOS+$2A
xBUFSIZE equ xBIOS+$3f1           ; Buffer size lo byte $100-SIZE (
xDEVICE equ xBIOS+$3fc           ; Device ID
xIOV equ xBIOS+$3ee             ; I/O module entry (2 bytes)

xBSIO    jmp     (xIOV)
start    jsr     xBIOS_SET_DEFAULT_DEVICE ; I want to use xB SIO I/O
        lda     #$100-$01           ; set buffer size
        sta     xBUFSIZE
        lda     #'1'                ; AtariOS device '1' = DOS device
        sta     xDEVICE             ; you can use '2', '3' and so for
        ldx     #$3F                ; set command GET HI SPEED FROM D
        jsr     xBSIO
        bcs     DRIVE_HAS_NO_HIGH_SPEED_FUNCTIONS

```

```

lda      $7ff                ; get HiSpeedIndex byte from buff
                                ; (xBUFSIZE = 10 byte,xBUFFERH =

```

Remember to preserve old xBUFSIZE and xDEVICE values if these are required later in your code.

Using Alternative Disk Drives#

If you are using disk drives other than the default D1:, you can save values into xDEVICE in order to refer to the other disk drives.

```

xDEVICE equ xBIOS+$3fc ; Device ID

lda #2                ; Use disk drive number 2
sta xDEVICE

```

Checking if Alternative Disk Drives Exist#

Apply this code to check if a particular disk drive exists (1-8).

```

lda #device_number
sta xDEVICE
jsr status

```

Varying Directory Sizes#

xBIOS can handle directories of varying sizes as per the directories of different versions of DOS. ? xDIRSIZE? (xBIOS+\$3e5 in xBIOS 4.3) is used as a read only way to find out the current directory size in sectors. It is one byte in length and is read-only.

A directory entry within Atari DOS 2 / MyDOS and similar DOS?s uses up 16 bytes and looks like this:

```

.byte status          ; Byte 1
.word size            ; - in standard directories always $08 ; Bytes 2 and 3
.word first_sector   ; Bytes 4 and 5
.byte c'FILENAMEEXT' ; Byte 6 to 16

```

A standard directory uses 8 sectors (size = \$08) which means 8 sectors * 8 entry = 64 max. Top DOS / Bibo DOS with sector sizes of 256 bytes may look like 8 sectors * 16 entry = 128 maximum. You can make subdirectories of differing sizes eg. size=\$02 (16/32 entry) or size = \$ff (2040 / 4080 entry). There is no tool to make different size catalogues, if at any time such a tool appears, xBIOS will handle it.

xBOOT#

This isn't xBIOS as such but a related loader mechanism which is simpler than xBIOS. Simply download xBOOT from <http://xxl.atari.pl> and add it to your disk using an external tool such as ? franny?. Name the first file to load ?AUTORUN?. This will auto-load. Within this executable, have the following code to load the next file in a chain.

```

xBOOT_LOAD_FILE equ $5f1

ldy <fname
ldx >fname
jmp xBOOT_LOAD_FILE
fname .byte c'NEXTPARTCOM' ; 11 chars ATASCII

```

xBOOT uses 384 bytes at \$480 and \$f9-\$ff in page zero.

Original document from Steve Nicklin (snicklin at AtariAge).

Imported into AtariWiki (23/01/2016) to allow peer feedback and improvement.