NextOS API (Updated 28 Jan 2018)

This document describes the **NextOS API**, which directly descends from the **+3DOS API** present in the *Sinclair ZX Spectrum +2A/+2B/+3* and the **IDEDOS API** additionally provided with the *ZX Spectrum +3e* ROMs.

It also describes the provided **esxDOS**-compatible API, which is compatible with esxDOS 0.8.x, but contains several enhancements.

This should be read in conjunction with the other documents:

- NextBASIC file-related commands and features
- NextBASIC new commands and features
- NextOS Editor features
- NextOS Unimplemented features

A list of updates made to this document is now provided at the end.

Available APIs

NextOS provides 2 distinct and separate APIs:

- a **+3DOS**-compatible API, providing the main **NextOS** API
- an **esxDOS**-compatible API, providing file-based calls for SD card access

The **+3DOS**-compatible API descends directly from the original +3DOS, provided with the Sinclair ZX Spectrum +3/+2A/+2B.

The **esxDOS**-compatible API is provided by a thin layer on top of +3DOS, and is compatible with esxDOS 0.8.x, with some additional facilities such as support for long filenames (LFNs), wildcards in filenames, enhanced dot command features and a low-overhead file streaming facility.

Both APIs provide general file-access calls. The **esxDOS**-compatible API is generally easier to use, but lacks the ability to access files on filesystems which are not FAT16/32 (such as the RAMdisk, and mounted CP/M and +3 disk images). It also lacks some of the more advanced features of the **+3DOS**-compatible API, such as bank allocation, BASIC command execution and file-browser dialogs.

The **+3DOS-**compatible API is described in the first section of the following pages, with the **esxDOS-**compatible API described in second section.

IMPORTANT NOTE:

When calling either the **+3DOS**-compatible or **esxDOS**-compatible API, make sure you have not left layer 2 writes enabled (ie bit 0 of port $123b should be zero when making any API call).

This is important because if layer 2 writes are left enabled, they can interfere with the operation of the system calls, which page in DivMMC RAM to the same region of memory ($0000-$3fff).

It is perfectly okay to leave layer 2 turned on and displayed (with bit 1 of port $123b) during API calls; only the writes need to be disabled.

The +3DOS-compatible API

The **+3DOS**-compatible API provides most of the facilities available on both the original +3/+2A/+2B, and the later +3e ROMs, with many additional facilities specific to the Next.

To make a +3DOS API call, you must first ensure that the memory bank configuration is set up correctly (with ROM 2 selected at the bottom of memory, RAM bank 7 at the top of memory and the stack located below $BFE0).

Once this is done, call the address indicated in the API call. You then probably want to restore the memory configuration to normal (with ROM 3 selected at the bottom of memory, and RAM bank 0 at the top of memory).

Please note that a few calls require the memory configuration to be slightly different on entry (with RAM bank 0 at the top of memory); this is noted in the individual documentation for those calls, which are generally BASIC-releated (eg IDE_STREAM_* and IDE_BASIC).

Useful example code showing how to use the API is available in the original +3 manual (section “Calling +3DOS from BASIC”), online here:

http://www.worldofspectrum.org/ZXSpectrum128+3Manual/chapter8pt26.html

This document does not describe unchanged calls, which are available in these online documents:

http://www.worldofspectrum.org/ZXSpectrum128+3Manual/chapter8pt27.html
http://www.worldofspectrum.org/zxplus3e/idedos.html

The following filesystem-related API calls are provided (*=effects have changed since originally documented in +3 manual or on +3e website; %=new for **NextOS**):

DOS_VERSION ($0103)	Get +3DOS issue and version numbers
*DOS_OPEN ($0106)	Create and/or open a file
DOS_CLOSE ($0109)	Close a file
DOS_ABANDON ($010C)	Abandon a file
DOS_REF_HEAD ($010F)	Point at the header data for this file
DOS_READ ($0112)	Read bytes into memory
DOS_WRITE ($0115)	Write bytes from memory
DOS_BYTE_READ ($0118)	Read a byte
DOS_BYTE_WRITE ($011B)	Write a byte
*DOS_CATALOG ($011E)	Catalog disk directory
*DOS_FREE_SPACE ($0121)	Free space on disk
DOS_DELETE ($0124)	Delete a file
DOS_RENAME ($0127)	Rename a file
DOS_BOOT ($012A)	Boot an operating system or other program
DOS_SET_DRIVE ($012D)	Set/get default drive
DOS_SET_USER ($0130)	Set/get default user number
*DOS_GET_POSITION ($0133)	Get file pointer for random access
DOS_SET_POSITION ($0136)	Set file pointer for random access
*DOS_GET_EOF ($0139)	Get end of file position for random access
DOS_GET_1346 ($013C)	Get memory usage in pages 1, 3, 4, 6
DOS_SET_1346 ($013F)	Re-allocate memory usage in pages 1, 3, 4, 6
DOS_FLUSH ($0142)	Bring disk up to date
DOS_SET_ACCESS ($0145)	Change open file's access mode
DOS_SET_ATTRIBUTES ($0148)	Change a file's attributes
DOS_SET_MESSAGE ($014E)	Enable/disable error messages
IDE_VERSION ($00A0)	Get IDEDOS version number
IDE_SWAP_OPEN ($00D9)	Open a swap partition
IDE_SWAP_CLOSE ($00DC)	Close a swap partition
IDE_SWAP_OUT ($00DF)	Write block to swap partition
IDE_SWAP_IN ($00E2)	Read block from swap partition
IDE_SWAP_EX ($00E5)	Exchange block with swap partition
IDE_SWAP_POS ($00E8)	Get current block number in swap partition
IDE_SWAP_MOVE ($00EB)	Set current block number in swap partition
IDE_SWAP_RESIZE ($00EE)	Change block size of swap partition
IDE_PARTITION_FIND ($00B5)	Find named partition
*IDE_DOS_MAP ($00F1)	Map drive to partition
*IDE_DOS_UNMAP ($00F4)	Unmap drive
IDE_DOS_MAPPING ($00F7)	Get drive mapping
*IDE_SNAPLOAD ($00FD)	Load a snapshot
*IDE_PATH ($01b1)	Create, delete, change or get directory
%IDE_CAPACITY ($01b4)	Get card capacity
%IDE_GET_LFN ($01b7)	Get long filename
%IDE_BROWSER ($01ba)	File browser

The following non-filesystem-related API calls are provided:

IDE_STREAM_OPEN ($0056)	Open stream to a channel
IDE_STREAM_CLOSE ($0059)	Close stream and attached channel
IDE_STREAM_IN ($005c)	Get byte from current stream
IDE_STREAM_OUT ($005f)	Write byte to current stream
IDE_STREAM_PTR ($0062)	Get or set pointer information for current stream
%IDE_BANK ($01bd)	Allocate or free 8K banks in ZX or DivMMC memory
%IDE_BASIC ($01c0)	Execute a BASIC command line
%IDE_WINDOW_LINEIN ($01c3)	Input line from current window stream
%IDE_WINDOW_STRING ($01c6)	Output string to current window stream
%IDE_INTEGER_VAR ($01c9)	Get or set NextBASIC integer variable
%IDE_RTC ($01cc)	Query the real-time-clock module
%IDE_DRIVER ($01cf)	Access the driver API

The following API calls are related to floppy drives and will not be useful for most software (included for legacy software use only):

DOS_REF_XDPB ($0151)	Point at XDPB for low level disk access
DOS_MAP_B ($0154)	Map B: onto unit 0 or 1
DD_INTERFACE ($0157)	Is the floppy disk driver interface present?
DD_INIT ($015A)	Initialise disk driver
DD_SETUP ($015D)	Specify drive parameters
DD_SET_RETRY ($0160)	Set try/retry count
DD_READ_SECTOR ($0163)	Read a sector
DD_WRITE_SECTOR ($0166)	Write a sector
DD_CHECK_SECTOR ($0169)	Check a sector
DD_FORMAT ($016C)	Format a track
DD_READ_ID ($016F)	Read a sector identifier
DD_TEST_UNSUITABLE ($0172)	Test media suitability
DD_LOGIN ($0175)	Log in disk, initialise XDPB
DD_SEL_FORMAT ($0178)	Pre-initialise XDPB for DD FORMAT
DD_ASK_1 ($017B)	Is unit 1 (external drive) present?
DD_DRIVE_STATUS ($017E)	Fetch drive status
DD_EQUIPMENT ($0181)	What type of drive?
DD_ENCODE ($0184)	Set intercept routine for copy protection
DD_L_XDPB ($0187)	Initialise an XDPB from a disk specification
DD_L_DPB ($018A)	Initialise a DPB from a disk specification
DD_L_SEEK ($018D)	uPD765A seek driver
DD_L_READ ($0190)	uPD765A read driver
DD_L_WRITE ($0193)	uPD765A write driver
DD_L_ON_MOTOR ($0196)	Motor on, wait for motor-on time
DD_L_T_OFF_MOTOR ($0199)	Start the motor-off ticker
DD_L_OFF_MOTOR ($019C)	Turn the motor off

The following API calls are present but generally for system use only and not useful for games/applications:

DOS_INITIALISE ($0100)	Initialise +3DOS
IDE_INTERFACE ($00A3)	Initialise card interfaces
IDE_INIT ($00A6)	Initialise IDEDOS
IDE_DRIVE ($00A9)	Get unit handle
*IDE_SECTOR_READ ($00AC)	Low-level sector read
*IDE_SECTOR_WRITE ($00AF)	Low-level sector write
*IDE_PARTITION_NEW ($00B8)	Create partition
*IDE_PARTITION_INIT ($00BB)	Initialise partition
IDE_PARTITON_READ ($00C4)	Read a partition entry
IDE_PARTITION_OPEN ($00CD)	Open a partition
IDE_PARTITION_CLOSE ($00D0)	Close a partition
IDE_PARTITIONS ($01a5)	Get number of open partitions

The following API calls were previously available in +3DOS/IDEDOS but are now deprecated and will return an error of rc_notimp:

DOS_OPEN_DRIVE ($014B)	Open a drive as a single file
IDE_FORMAT ($00B2)	Format a partition
IDE_PARTITION_ERASE ($00BE)	Delete a partition
IDE_PARTITION_RENAME ($00C1)	Rename a partition
IDE_PARTITION_WRITE ($00C7)	Write a partition entry
IDE_PARTITION_WINFO ($00CA)	Write type-specific partition information
IDE_PARTITION_GETINFO ($00D3)	Get byte from type-specific partition information
IDE_PARTITION_SETINFO ($00D6)	Set byte in type-specific partition information
IDE_DOS_UNPERMANENT ($00FA)	Remove permanent drive mapping

IDE_IDENTIFY ($01a2) Return IDE drive identity information

Updated calls

The following calls have new/updated features, which are highlighted in GREEN. (Some changes are due to removed parameters which are not shown). **NOTE:** Calls for internal use only have not yet been included here.

It should additionally be noted that the **IDE_STREAM_*** calls may corrupt the alternate register set, in addition to the effects on the standard register set noted for each individual call.

As well as describing additional features, DOS_CATALOG contains additional text which clarifies points that are not obvious from the documentation in the original +3 manual.

DOS_OPEN
0106h (262)

Create and/or open a file

There is a choice of action depending on whether or not the file already exists. The choices are 'open action' or 'create action', and are specified in DE. If the file already exists, then the open action is followed; otherwise the create action is followed.

Open action

0. Error - File already exists.

1. Open the file, read the header (if any). Position file pointer after header.

2. Open the file, ignore any header. Position file pointer at 000000h (0).

3. Assume given filename is 'filename.type'. Erase 'filename.BAK' (if it exists). Rename 'filename.type' to 'filename.BAK'. Follow create action.

4. Erase existing version. Follow create action.

Create action

0. Error - File does not exist.

1. Create and open new file with a header. Position file pointer after header.

2. Create and open new file without a header. Position file pointer at 000000h (0).

(Example: To simulate the tape action of... 'if the file exists open it, otherwise create it with a header', set open action = 1, create action = 1.)

(Example: To open a file and report an error if it does not exist, set open action = 1, create action = 0.)

(Example: To create a new file with a header, first renaming any existing version to '.BAK', set open action = 3, create action = 1.)

Files with headers have their EOF position recorded as the smallest byte position greater than all written byte positions.

Files without headers have their EOF position recorded as the byte at the start of the smallest 128 byte record position greater than all written record positions.

Soft-EOF is the character 1Ah (26) and is nothing to do with the EOF position, only the routine DOS BYTE READ knows about soft-EOF.

The header data area is 8 bytes long and may be used by the caller for any purpose whatsoever. If open action = 1, and the file exists (and has a header), then the header data is read from the file, otherwise the header data is zeroised. The header data is available even if the file does not have a header. Call DOS REF HEAD to access the header data.

Note that +3 BASIC makes use of the first 7 of these 8 bytes as follows:

BYTE	0	1	2	3	4	5	6
Program	0	file length		8000h or LINE		offset to prog	
Numeric array	1	file length		xxx	name	xxx	xxx
Character array	2	file length		xxx	name	xxx	xxx
CODE or SCREEN$	3	file length		load address		xxx	xxx

(xxx = doesn't matter)

If creating a file that will subsequently be LOADed within BASIC, then these bytes should be filled with the relevant values.

If the file is opened with exclusive-write or exclusive-read-write access (and the file has a header), then the header is updated when the file is closed.

A file that is already open for shared-read access on another file number may only be opened for shared-read access on this file number.

A file that is already open for exclusive-read or exclusive-write or exclusive-read-write access on another file number may not be opened on this file number.

If the open action is 1 or 2 and the create action is 0 (ie only an existing file is to be opened) then the filename may optionally contain the wildcard characters * and ?. In this case, the first file that matches the wildcard will be opened.

ENTRY CONDITIONS
        B = File number 0...15
        C = Access mode required
                Bits 0...2 values:
                        1 = exclusive-read
                        2 = exclusive-write
                        3 = exclusive-read-write
                        5 = shared-read
                Bits 3...7 = 0 (reserved)
        D = Create action
        E = Open action

```
        HL = Address of filename (no wildcards, unless D=0 and E=1 or 2)

EXIT CONDITIONS
        If file newly created:
                Carry true
                Zero true
                A corrupt
        If existing file opened:
                Carry true
                Zero false
                A corrupt
        Otherwise:
                Carry false
                A = Error code
        Always:
                BC DE HL IX corrupt
                All other registers preserved
```

DOS_CATALOG
011Eh (286)

Fills a buffer with part of the directory.

The filename optionally specifies the drive, path, user and a (possibly ambiguous) filename (which may contain wildcard characters ? and *).

Since the size of a directory is variable (and may be quite large), this routine permits the directory to be catalogued in a number of small sections. The caller passes a buffer pre-loaded with the first required filename, or zeroes for the start of the directory. The buffer is loaded with part (or all, if it fits) of the directory sorted in ASCII order. If more of the directory is required, this routine is re-called with the buffer re-initialised with the last file previously returned. This procedure is followed repeatedly until all of the directory has been catalogued.

Note that +3DOS format disks (which are the same as single-sided, single track AMSTRAD PCW range format disks) may have a maximum of 64 directory entries.

Buffer format:

```
        Entry 0
        Entry 1
        Entry 2
        Entry 3
        ...to...
        Entry n
```

Entry 0 must be preloaded with the first 'filename.type' required. Entry 1 will contain the first matching filename greater than the preloaded entry (if any). A zeroised preload entry is OK.

If the buffer is too small for the directory, this routine can be called again with entry 0 replaced by entry n to fetch the next part of the directory.

Entry format (13 bytes long):

```
        Bytes 0...7     - Filename (ASCII) left justified, space
```

```
                        filled
        Bytes 6...10    - Type (ASCII) left justified, space filledd
        Bytes 11...12   - Size in kilobytes (binary)
```

Any of the filename or extension characters may have bit 7 set, as described in the section on file attributes, so these should be masked off if not required.

The file size is the amount of disk space allocated to the file, not necessarily the same as the amount used by the file.

```
ENTRY CONDITIONS
        B = n+1, size of buffer in entries, >=2
        C = Filter (if bit is set)
                bit 0 = include system files
                bit 1 = set bit 7 of f7 (the 7th character in the filename) if
                        the entry has a valid LFN (long filename) which can be
                        obtained with the IDE_GET_LFN call
                bit 2 = include directories, and set bit 7 of f8 (the 8th
                        character in the filename) if the entry is a directory
                bits 3...7 = 0 (reserved)
        DE = Address of buffer (first entry initialised)
        HL = Address of filename (wildcards permitted)

EXIT CONDITIONS
        If OK:
                Carry true
                A corrupt
                B = Number of completed entries in buffer, 0...n.
                    (If B = n, there may be more to come).
                HL = Directory handle, required to obtain long filenames
                     with IDE_GET_LFN

        Otherwise:
                Carry false
                A = Error code
                B HL corrupt
        Always:
                C DE HL IX corrupt
                All other registers preserved
```

DOS_FREE_SPACE
0121h (289)

How much free space is there on this drive?

```
ENTRY CONDITIONS
        A = Drive, ASCII 'A'...'P'

EXIT CONDITIONS
        If OK:
                Carry true
                A corrupt
                HL = Free space (in kilobytes, clamped to maximum 65535K)
                BCDE = Free space (in kilobytes)

        Otherwise:
                Carry false
                A = Error code
                HL corrupt
```

```
        Always:
                BC DE IX corrupt
                All other registers preserved
```

DOS_GET_POSITION
0133h (307)

Get the file pointer.

```
ENTRY CONDITIONS
        B = File number

EXIT CONDITIONS
        If OK:
                Carry true
                A corrupt
                DEHL = File pointer
                (D holds most significant byte; L holds least
                significant byte)
        Otherwise:
                Carry false
                A = Error code
                DE HL corrupt
        Always:
                BC IX corrupt
                All other registers preserved
```

DOS_GET_EOF
0139h (313)

Get the end of file (EOF) file position greater than all written byte positions.

Does not affect the file pointer.

Does not consider soft-EOF.

```
ENTRY CONDITIONS
        B = File number

EXIT CONDITIONS
        If OK:
                Carry true
                A corrupt
                DEHL = File pointer
                (D holds most significant byte; L holds least
                significant byte)
        Otherwise:
                Carry false
                A = Error code
                DE HL corrupt
        Always:
                BC IX corrupt
                All other registers preserved
```

IDE_DOS_MAP ($00F1)

Map a drive to the specified partition or physical device

```
IN:  A=unit (0 or 1), or physical device:
                    2=floppy device 0
                    3=floppy device 1
                    4=RAMdisk
     BC=partition number
     L=drive letter 'A' to 'P' (uppercase)

OUT(s):  Fc=1
OUT(f):  Fc=0, A=error code

Register status on return:
........./IX same
AFBCDEHL/.. different
```

IDE_DOS_UNMAP ($00F4)

Remove mapping from the specified drive

```
IN:  L=drive letter 'A' to 'P' (uppercase)

OUT(s):  Fc=1
OUT(f):  Fc=0, A=error code

Register status on return:
........./IX same
AFBCDEHL/.. different
```

IDE_SNAPLOAD ($00FD)

Load a snapshot

```
IN:  HL=filespec, terminated with $ff

OUT(s):  Does not return if successful
OUT(f):  Fc=0, A=error code

Register status on return:
......../.. same
AFBCDEHL/IX different
```

Loads and runs a supported snapshot file (files with extension .Z80, .SNA, .O and .P are supported, with others potentially supported in future).

IDE_PATH ($01b1)

```
IN:  A=reason code,
      rc_path_change (0),
      rc_path_get (1),
      rc_path_make (2),
      rc_path_delete (3)

    HL=address of pathspec (terminated with $ff)
       NB: For rc_path_get, this must also be a 256-byte buffer
```

into which the returned path will be written

OUT(s): Fc=1
OUT(f): Fc=0, A=error code

Register status on return:
...../.... same
AFBCDEHL/IXIY different

This call allows the current directory or path for a particular drive (and user area) to be changed or obtained. It also allows creation and deletion of directories.

For rc_path_change, rc_path_make and rc_path_delete, HL points to a directory specification, terminated by $ff. This may optionally include a drive letter, user area and full path (if not, the current default values are used). For rc_path_change, the current path on that drive is changed to the directory or path specified. For rc_path_make and rc_path_delete, the named directory is created or deleted.

For rc_path_get, HL points to a location specification (ie a drive and/or user area, terminated with a colon and $ff). The current path for that location will then be written to the buffer at HL and terminated with $ff.

Note that this call will return an error of rc_notimp if the drive on which it is operating is formatted with a filesystem that does not support directories (eg a +3DOS floppy drive or RAMdisk).

New calls

The following calls are new for **NextOS.**

IDE_CAPACITY ($01b4)

Get card capacity

IN: C=unit (0 or 1)

OUT(s): Fc=1
DEHL=total card capacity in 512-byte sectors
OUT(f): Fc=0, A=error code

Register status on return:
...../.. same
AFBCDEHL/IX different

IDE_GET_LFN ($01b7)

Obtain a long filename and other file information

IN: HL=address of filespec provided to the last **DOS_CATALOG** call
IX=directory handle returned by the last **DOS_CATALOG** call
DE=address of a file entry within buffer filled by the last **DOS_CATALOG** call
BC=address of a 261-byte buffer to receive the long filename

OUT(s): Fc=1
Buffer at BC is filled with the long filename for the requested entry, terminated with $ff. If no long filename was available, the buffer will contain the properly-formatted short filename instead.
BC=date (in MS-DOS format)
DE=time (in MS-DOS format)
HLIX=filesize (in bytes)
OUT(f): Fc=0, A=error code

Register status on return:
...../.. same
AFBCDEHL/IX different

This call allows a long filename (or properly-formatted short filename) for an entry in the buffer returned by **DOS_CATALOG** to be obtained. It also returns additional directory entry details (date, time, file size).

NOTE: No other +3DOS calls should be made between the **DOS_CATALOG** call and the (multiple) **IDE_GET_LFN** calls used to obtain the long filenames.

NOTE: If the file entry is a directory, the filesize returned in HLIX will be zero.

IDE_BROWSER ($01ba)

Run the file browser

IN: HL=address of supported filetypes buffer, laid out as follows:
+0 (1 byte) Length of next entry, n

```
          +1 (n bytes) 1-3 byte extension, colon, optional BASIC command(s)
        If n=$ff there are no further entries.
     DE=address of $ff-terminated help text for 2 lines at bottom of screen
     A=browser capabilities mask, made by ORing together any of:
            $01, BROWSERCAPS_COPY   - files may be copied
            $02, BROWSERCAPS_RENAME - files/dirs may be renamed
            $04, BROWSERCAPS_MKDIR  - directories may be created
            $08, BROWSERCAPS_ERASE  - files/dirs may be erased
            $10, BROWSERCAPS_REMOUNT- SD card may be remounted
            $80, BROWSERCAPS_SYSCFG - system use only - use browser.cfg
     Alternatively just use one of the two special values:
            $00, BROWSERCAPS_NONE   - no special capabilities
            $1f, BROWSERCAPS_ALL    - all capabilities enabled

OUT(s):  Fc=1
         If Fz=1, ENTER was pressed with a filetype that is present in the
                  filetype buffer, and:
                  HL=address of short filename (terminated with $ff) in RAM 7
         If Fz=0, SPACE/BREAK was pressed
OUT(f):  Fc=0, A=error

Register status on return:
......../.. same
AFBCDEHL/IX different

NOTES:

The help text can contain any standard full-screen mode window control codes,
but if the character size is changed, it should be changed back to size 5 at the
end.

It is intended that applications wishing to use the Browser as a "save file"
dialog should direct the user to navigate to the correct drive/directory and
press SPACE. At this point the call will exit with the current drive and
directory set as the user selected and Fz=0 to indicate SPACE was pressed. Since
the screen is not cleared on exit, the application can then request input of the
filename on the bottom two lines of the screen, giving a seamless user
experience.

Call does not return if a supported filetype was selected which had anything
following the colon in the filetype buffer. In this case, the additional data is
treated as plain text, then tokenized and executed as a BASIC command. NOTE: No
terminator should be added to the end of the command.

The ? character may be used as a wildcard to match a single character in the
filetype.

The * character may be used as a wildcard to match remaining characters in the
filetype.

Most applications will not want a BASIC command to be executed and so should
provide a simple list of all the filetypes that they want to be selectable.

Example filetype buffer contents:
      defb  4                 ; length of first entry
      defm  "XYZ:"            ; match this filetype and return to caller with it
      defb  12                ; length of second entry
      defm  "X:.hexdump |"    ; match this filetype and execute .hexdump on it
      defb  3                 ; length of third entry
      defm  "Z?:"             ; matches .z3, .z4, .z5 etc
      defb  3                 ; length of fourth entry
      defm  "Z*:"             ; matches .z, .zip etc
```

```
      defb  $ff               ; table terminator
```

To match all files, you can provide a simple table like this:

```
      defb  2
      defm  "*:"
      defb  $ff
```

IDE_BANK ($01bd)

Allocate or free 8K RAM banks in main ZX memory or DivMMC memory

```
IN:  H=bank type:
        rc_banktype_zx (0), ZX memory half-banks (8K size)
        rc_banktype_mmc (1), DivMMC memory banks (8K size)
     L=reason:
        rc_bank_total (0), return total number of 8K banks of specified type
        rc_bank_alloc (1), allocate next available 8K bank
        rc_bank_reserve (2), reserve bank specified in E (0..total-1)
        rc_bank_free (3), free bank specified in E (0..total-1)
     E=8K bank ID (0..total-1), for rc_bank_reserve/rc_bank_free

OUT(s):  Fc=1
         E=8K bank ID (0..total-1), for rc_bank_alloc
         E=total number of 8K banks of specified type, for rc_bank_total

OUT(f):  Fc=0
         A=error: rc_inuse if no available banks to allocate
                  rc_badparam if H, L or E is invalid

Register status on return:
......../.. same
AFBCDEHL/IX different
```

NOTE:

This call is provided for applications that wish to co-exist with other applications, dot commands and BASIC programs without overwriting each other's memory.

Bank IDs are for 8K half-banks, numbered from 0 upwards. For ZX memory they can be paged using the MMU instructions.

NextOS/NextBASIC normally reserves the first 18 x 8K banks of ZX memory for its own use, and the first 6 x 8K banks of DivMMC memory. However, BASIC programs or TSR machine code programs could also reserve memory before your program is loaded, so it is usually easier to allocate using rc_bank_alloc rather than rc_bank_reserve.

NextOS/NextBASIC also owns the layer 2 banks (normally 16K banks 9,10,11: 8K banks 18-23, but may have been changed by the LAYER BANK command). However, you can use such banks if you are in control of the system and not using layer 2: the current layer 2 banks can be found by reading Next registers $12 and $13 to find the base of the current front and back buffers, respectively.

Take care to free any banks you allocate before exiting, otherwise they will be unavailable to the user until after a reset. A NEW command *does not* free reserved banks back into the system.

IDE_BASIC ($01c0)

Execute a BASIC command line

IN: HL=address of tokenized BASIC command line, terminated with $0d

OUT(s): Fc=1
         System variable ERR_NR contains generated BASIC error code-1
         ($ff means BASIC command completed successfully)

Register status on return:
......../.. same
AFBCDEHL/IX different

NOTES:

This call must be made with the ROM2/RAM5/RAM2/RAM0 memory configuration rather than the usual +3DOS configuration. The stack must be located between STKEND and RAMTOP (the normal location for the stack during BASIC operation).

Any number of BASIC commands may be executed, separated by colons (:), and the line must be terminated with an ENTER character ($0d).

This call may be particularly useful for setting particular screen modes with the LAYER command, which will ensure that the system variables are correctly set up for printing to windows or the main screen in the selected mode.

IDE_WINDOW_LINEIN ($01c3)

Input line from current window stream

IN: required window has been made current via ROM 3 / $1601
     HL=buffer address (must lie entirely below $c000)
     A=buffer size (1..255 bytes)
     E=number of characters already in the input buffer (0 for an entirely new input). Must be less than A.

OUT: E=number of characters returned in input buffer

Register status on return:
......../.. same
AFBCDEHL/IX different

NOTES:

This call invokes the window line input handler, allowing the user to enter new characters and edit the input with the cursor keys and delete.

The input buffer can be primed with an initial string for the user to edit. If this is the case, E should be set to the number of characters in the initial string (otherwise, set E=0).

+3 BASIC errors may be invoked

IDE_WINDOW_STRING ($01c6)

Output string to current window stream

```
IN:  required window has been made current via ROM 3 / $1601
     HL=address of string (must lie entirely below $c000)
     E=string termination condition:
       if E=$ff, string is terminated with a $ff character
       if E=$80, last character in the string has bit 7 set
       if E<$80, E=number of characters in the string (may be
                 terminated earlier with $ff)
OUT: -

Register status on return:
........../.. same
AFBCDEHL/IX different

NOTES:

This call is intended for efficient outputting of strings to window channels,
avoiding the significant per-character overhead associated with outputting each
individual character via RST $10 or IDE_STREAM_OUT.

+3 BASIC errors may be invoked
```

IDE_INTEGER_VAR ($01c9)

```
Get or set NextBASIC integer variable

IN:  B=0 for standard variable, B=1 for array
     C=variable number (0=A,1=B...25=Z)
     L=array index (0..63) if B=1
     H=0 to get variable, 1 to set variable
     DE=value (if H=1)

OUT(s):  Fc=1
         DE=value (if H=0)

OUT(f):  Fc=0
         A=error: rc_badparam if H, L or E is invalid

Register status on return:
........../.. same
AFBCDEHL/IX different

NOTE:

This call provides a convenient interface to pass values between BASIC and
machine-code processes.
```

IDE_RTC ($01cc)

```
Query the real-time-clock module

IN:  -

OUT(s):  Fc=1
         BC=date, in MS-DOS format
         DE=time, in MS-DOS format

OUT(f):  Fc=0, real-time-clock module not present
```

Register status on return:
...../.. same
AFBCDEHL/IX different

NOTE:

This call returns the results provided by the RTC.SYS loadable module.

IDE_DRIVER ($01cf)

Access the driver API

IN: C=driver id
     B=call id
     HL,DE=other input parameters as described in driver API

OUT(s): Fc=1
         Other results as described in M_DRVAPI

OUT(f): Fc=0, error
         Other results as described in M_DRVAPI

Register status on return:
...../.. same
AFBCDEHL/IX different

NOTE:

This call is equivalent to the M_DRVAPI hook provided in the esxDOS API. Applications will probably find M_DRVAPI more convenient to use; this call is designed for use by the NextOS ROMs.

This call should be made with the ROM2/RAM5/RAM2/RAM0 memory configuration rather than the usual +3DOS configuration.

HL is used as an input value instead of IX (ie same as calling M_DRVAPI from a dot command).

Error codes

The error codes that may be returned by +3DOS/IDEDOS calls are as follows:
Recoverable disk errors:

0	rc_ready	Drive not ready
1	rc_wp	Disk is write protected
2	rc_seek	Seek fail
3	rc_crc	CRC data error
4	rc_nodata	No data
5	rc_mark	Missing address mark
6	rc_unrecog	Unrecognised disk format
7	rc_unknown	Unknown disk error
8	rc_diskchg	Disk changed whilst +3DOS was using it
9	rc_unsuit	Unsuitable media for drive

Non-recoverable errors:

20	rc_badname	Bad filename
21	rc_badparam	Bad parameter
22	rc_nodrive	Drive not found
23	rc_nofile	File not found
24	rc_exists	File already exists
25	rc_eof	End of file
26	rc_diskfull	Disk full
27	rc_dirfull	Directory full
28	rc_ro	Read-only file
29	rc_number	File number not open (or open with wrong access)
30	rc_denied	Access denied
31	rc_norename	Cannot rename between drives
32	rc_extent	Extent missing
33	rc_uncached	Uncached
34	rc_toobig	File too big
35	rc_notboot	Disk not bootable
36	rc_inuse	Drive in use
56	rc_invpartition	Invalid partition
57	rc_partexist	Partition already exists
58	rc_notimp	Not implemented
59	rc_partopen	Partition open
60	rc_nohandle	Out of handles
61	rc_notswap	Not a swap partition
62	rc_mapped	Drive already mapped
63	rc_noxdpb	No XDPB
64	rc_noswap	No suitable swap partition
65	rc_invdevice	Invalid device
67	rc_cmdphase	Command phase error
68	rc_dataphase	Data phase error
69	rc_notdir	Not a directory

The esxDOS-compatible API

The esxDOS-compatible API is a bit simpler to use than the +3DOS-compatible API.

To make a call, you only need to set up the entry parameters as indicated and perform a **RST $08; DEFB hook_code**. On return, registers AF,BC,DE,HL will all be changed. IX,IY and the alternate registers are never changed (except for **M_P3DOS**).

(Note that the standard 48K BASIC ROM must be paged in to the bottom of memory, but this is the usual situation after starting a machine code program with a **USR** function call).

Notice that error codes are different from those returned by +3DOS calls, and also the carry flag is SET for an error condition when returning from an esxDOS call (instead of RESET, as is the case for +3DOS).

If desired, you can use the **M_GETERR** hook to generate a BASIC error report for any error returned, or even use it to generate your own custom BASIC error report.

All of the calls where a filename is specified will accept long filenames (LFNs) and most will accept wildcards (for an operation such as F_OPEN where a single file is always used, the first matching filename will be used).

Dot commands

Dot commands can also be written using the esxDOS-compatible API. Normally dot commands run from the C:/BIN/ directory, but they can be run from anywhere if fully-pathed. For example:

```
.mydot          ; executes C:/BIN/mydot
./mydot         ; executes /mydot on current drive
../mydot        ; executes mydot from current directory on current drive
```

The default Browser configuration supports selecting and running dot commands if they have a .DOT extension.

Requirements

A dot command must be assembled to run at origin $2000, and will be loaded into DivMMC RAM to execute. The maximum code/data size available is 8K.

It is permissable to relocate the stack to within the 8K area if desired (except when calling an external ROM with **RST $10**, **RST $18** or the **M_P3DOS** hook code).

On entry to your dot command, HL contains the address of the arguments following the command name (or 0 if there are no arguments). Additionally, BC contains the address of the entire command line (including the command name but excluding the leading ".").

The arguments/command line may be terminated by $00, $0d or ':' (since the address usually points within a BASIC statement, but may also be a system-supplied null-terminated line).

On exit from your dot command, return with the the carry flag reset if execution was successful.

To report a standard esxDOS error, set the carry flag and return with A=error.

To generate a custom error report, set the carry flag and return with A=0 and HL=address of error message (last character must have bit 7 set).

Calling esxDOS-compatible API hooks

When called from within dot commands, the entry parameters used for **RST $8** hook codes are slightly different: HL should be used instead of IX. Exit parameters are unchanged.

Calling external ROM routines

Within dot commands, two further restarts are available to call routines in the standard 48K BASIC ROM:

RST $10
Print the character in A (NOTE: A must not be $80).

RST $18; DEFW address
Call any routine in the standard 48K BASIC ROM.

If a BASIC error occurs during a **RST $10** or **RST $18** call (eg the user presses BREAK at a "scroll?" prompt) the dot command will be terminated and the error reported, unless you have registered an error handler with the **M_ERRH** hook.

Large dot commands

If your dot command is >8K in length, only the first 8K is loaded (at $2000), but the file is left open (with the pointer directly after the first 8K). It is possible to obtain the file handle using the **M_GETHANDLE** hook. This allows you to read further code/data from your dot command into another memory area (perhaps a bank allocated using **IDE_BANK** via **M_P3DOS**) or into the standard 8K area as required.

Bootstrapping a game/application from a dot command

You can write large dot commands that load all the initial assets for a game/application into memory (probably in the way described for large dot commands above) and then start running them.

The recommended way to start your game/application after loading from within a dot command is to use **RST $20** with HL=address. This will cleanly terminate your dot command, and return to the address provided in HL.

Note that this still leaves your dot command file open (as well as any other files you may have opened), so you may continue to load further assets from it if desired.

NOTE:

Although it is possible to start your game/application by simply jumping to the code you have loaded (rather than using the **RST $20** mechanism), this is not recommended since doing so will leave the DivMMC ROM/RAM paged in place of the standard 48K BASIC ROM. The main disadvantages of this would be:

- writing to Next registers MMU0/1 will have no effect
- needing to continue to use RST $8 hooks as if the dot command was running
- inability to run any further dot commands
- standard IM1 interrupt routine (including ROM keyscanning) unavailable
- NMI unavailable, so Multiface replacement can't be activated

(NOTE: If you don't want your game to be interruptible/snapshottable by the Multiface replacement, this can be achieved anyway by clearing the multiface enable bit (bit 3) in the Next's peripheral2 register, $06).

Installable device drivers

NextOS allows for a number of drivers to be installed/uninstalled at will using the .install/.uninstall dot commands (currently a maximum of 4 drivers may be installed at any one time). These are mainly intended for use as drivers for external peripherals such as printers, mice, network devices etc, but could be used for other purposes.

Each driver occupies a maximum of 512 bytes, which is loaded into DivMMC RAM and relocated by the .install command. It is possible to allocate additional 8K banks of DivMMC RAM and/or standard ZX Spectrum Next RAM during installation if required (note that RAM is a limited resource).

Drivers have two entry points: an (optional) routine which is run during interrupts, and an API routine which allows the driver to respond to user requests. The driver's API is accessible from the **M_DRVAPI** hook (in the esxDOS-compatible API), the **IDE_DRIVER** call (in the +3DOS-compatible API) and the **DRIVER** command in *NextBASIC*.

Each driver is identified by a unique single-byte id, so when writing a new driver you should ensure that it's id does not clash with any other existing driver. However, it would be acceptable for multiple different drivers to all use the same identifier as long as they provide the same functionality via their APIs (for example, multiple drivers for different printer interfaces might all use the 'P' identifier).

Channel support

Drivers can optionally be written to support i/o via the streams and channels system of the Spectrum Next. This would allow the following BASIC commands to open and close streams to the device (it is up to your documentation to describe which of the **OPEN #** variants should be used):

OPEN #*n*,"**D>***X*"
open stream n to simple channel for device 'X'

OPEN #*n*,"**D>***X***>***string*"
open stream n to channel described by *string* on device 'X'

OPEN #n,"**D>***X*,*p1*"
open stream n to channel described by numeric value *p1* on device 'X'

OPEN #n,"**D>***X*,*p1*,*p2*"
open stream n to channel described by numeric values *p1* and *p2* on device 'X'

CLOSE #*n*
close stream n

Once a channel is open, devices can (optionally) accept any of stream input, output or pointer manipulation through their APIs which will allow other stream-related BASIC commands to be used, eg:

PRINT #*n;....*
INPUT #*n;....*
INKEY$ #*n*
RETURN #*n,var* (get current stream pointer to variable *var*)
DIM #*n,var* (get current stream size/extent to variable *var*)
GOTO #*n,value* (set current stream pointer)
NEXT #*n,var* (wait for next input character from stream and store in *var*)

For information on writing device drivers, see the worked example in border.asm and border_drv.asm (available separately or at the end of this document).

The following calls are available in the esxDOS-compatible API:

```
; Low-level calls

disk_filemap       ; $85 (133)          obtain file allocation map
disk_strmstart     ; $86 (134)          start streaming operation
disk_strmend       ; $87 (135)          end streaming operation

; Miscellaneous calls.

m_dosversion       ; $88 (136)          get NextOS version/mode information
m_getsetdrv        ; $89 (137)          get/set default drive
m_tapein           ; $8b (139)          tape redirection control (input)
m_tapeout          ; $8c (140)          tape redirection control (output)
m_gethandle        ; $8d (141)          get handle for current dot command
m_getdate          ; $8e (142)          get current date/time
m_execcmd          ; $8f (143)          execute a dot command
m_drvapi           ; $92 (146)          access API for installable drivers
m_geterr           ; $93 (147)          get or generate error message
m_p3dos            ; $94 (148)          execute +3DOS/IDEDOS/NextOS call
m_errh             ; $95 (149)          register dot command error handler

; File calls.

f_open             ; $9a (154)          open file
f_close            ; $9b (155)          close file
f_sync             ; $9c (156)          sync file changes to disk
f_read             ; $9d (157)          read file
f_write            ; $9e (158)          write file
f_seek             ; $9f (159)          set file position
f_fgetpos          ; $a0 (160)          get file position
f_fstat            ; $a1 (161)          get open file information
f_ftruncate        ; $a2 (162)          truncate/extend open file
f_opendir          ; $a3 (163)          open directory for reading
f_readdir          ; $a4 (164)          read directory entry
f_telldir          ; $a5 (165)          get directory position
f_seekdir          ; $a6 (166)          set directory position
f_rewinddir        ; $a7 (167)          rewind to start of directory
f_getcwd           ; $a8 (168)          get current working directory
f_chdir            ; $a9 (169)          change directory
f_mkdir            ; $aa (170)          make directory
f_rmdir            ; $ab (171)          remove directory
f_stat             ; $ac (172)          get unopen file information
f_unlink           ; $ad (173)          delete file
f_truncate         ; $ae (174)          truncate/extend unopen file
f_chmod            ; $af (175)          change file attributes
f_rename           ; $b0 (176)          rename/move file
f_getfree          ; $b1 (177)          get free space
```

esxDOS-compatible error codes

```
Unknown error                   ; 0, esx_ok
OK                              ; 1, esx_eok
Nonsense in esxDOS              ; 2, esx_nonsense
Statement end error             ; 3, esx_estend
Wrong file type                 ; 4, esx_ewrtype
No such file or dir             ; 5, esx_enoent
I/O error                       ; 6, esx_eio
Invalid filename                ; 7, esx_einval
Access denied                   ; 8, esx_eacces
Drive full                      ; 9, esx_enospc
Invalid i/o request             ; 10, esx_enxio
No such drive                   ; 11, esx_enodrv
Too many files open             ; 12, esx_enfile
Bad file number                 ; 13, esx_ebadf
No such device                  ; 14, esx_enodev
File pointer overflow           ; 15, esx_eoverflow
Is a directory                  ; 16, esx_eisdir
Not a directory                 ; 17, esx_enotdir
Already exists                  ; 18, esx_eexist
Invalid path                    ; 19, esx_epath
Missing system                  ; 20, esx_esys
Path too long                   ; 21, esx_enametoolong
No such command                 ; 22, esx_enocmd
In use                          ; 23, esx_einuse
Read only                       ; 24, esx_erdonly
Verify failed                   ; 25, esx_everify
Sys file load error             ; 26, esx_eloadingko
Directory in use                ; 27, esx_edirinuse
MAPRAM is active                ; 28, esx_emapramactive
Drive busy                      ; 29, esx_edrivebusy
Unknown filesystem              ; 30, esx_efsunknown
Device busy                     ; 31, esx_edevicebusy
```

```
; ***************************************************************************
; * DISK_FILEMAP ($85)                                                      *
; ***************************************************************************
; Obtain a map of card addresses describing the space occupied by the file.
; Can be called multiple times if buffer is filled, continuing from previous.
; Entry:
;       A=file handle (just opened, or following previous DISK_FILEMAP calls)
;       IX=buffer
;       DE=max entries (each 6 bytes: 4 byte address, 2 byte sector count)
; Exit (success):
;       Fc=0
;       DE=max entries-number of entries returned
;       HL=address in buffer after last entry
;       A=card flags: bit 0=card id (0 or 1)
;                     bit 1=0 for byte addressing, 1 for block addressing
; Exit (failure):
;       Fc=1
;       A=error
;
; NOTES:
; Each entry may describe an area of the file between 2K and just under 32MB
; in size, depending upon the fragmentation and disk format.
; Please see example application code, stream.asm, for full usage information
; (available separately or at the end of this document).


; ***************************************************************************
; * DISK_STRMSTART ($86)                                                    *
; ***************************************************************************
; Start reading from the card in streaming mode.
; Entry: IXDE=card address
;        BC=number of 512-byte blocks to stream
;        A=card flags
; Exit (success): Fc=0
;                 B=0 for SD/MMC protocol, 1 for IDE protocol
;                 C=8-bit data port
; Exit (failure): Fc=1, A=esx_edevicebusy
;
; NOTES:
; On the Next, this call always returns with B=0 (SD/MMC protocol) and C=$EB
; When streaming using the SD/MMC protocol, after every 512 bytes you must read
; a 2-byte CRC value (which can be discarded) and then wait for a $FE value
; indicating that the next block is ready to be read.
; Please see example application code, stream.asm, for full usage information
; (available separately or at the end of this document).


; ***************************************************************************
; * DISK_STRMEND ($87)                                                      *
; ***************************************************************************
; Stop current streaming operation.
; Entry: A=card flags
; Exit (success): Fc=0
; Exit (failure): Fc=1, A=esx_edevicebusy
;
; NOTES:
; This call must be made to terminate a streaming operation.
; Please see example application code, stream.asm, for full usage information
; (available separately or at the end of this document).
```

```
; ***************************************************************************
; * M_DOSVERSION ($88)                                                      *
; ***************************************************************************
; Get API version/mode information.
; Entry:
;       -
; Exit:
;       For esxDOS <= 0.8.6
;               Fc=1, error
;               A=14 ("no such device")
;
;       For NextOS:
;               Fc=0, success
;               B='N',C='X' (NextOS signature)
;               DE=NextOS version in BCD format: D=major, E=minor version number
;                                                eg for NextOS v1.94, DE=$0194
;               HL=A=0 if running in NextOS mode (and zero flag is set)
;               HL,A<>0 if running in 48K mode (and zero flag is reset)


; ***************************************************************************
; * M_GETSETDRV ($89)                                                       *
; ***************************************************************************
; Get or set the default drive.
; Entry:
;       A=0, get the default drive
;       A<>0, set the default drive to A
;             bits 7..3=drive letter (0=A...15=P)
;             bits 2..0=drive number (0)
; Exit (success):
;       Fc=0
;       A=default drive, encoded as:
;             bits 7..3=drive letter (0=A...15=P)
;             bits 2..0=drive number (0)
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTE:
; This call isn't really very useful, as it is not necessary to provide a
; specific drive to calls which need a drive/filename.
; For such calls, you can instead provide:
;   A='*'   use the default drive
;   A='$'   use the system drive (C:, where the NEXTOS and BIN directories are)
```

```
; ************************************************************************
; * M_TAPEIN ($8b)                                                       *
; ************************************************************************
; Tape input redirection control.
; Entry:
;       B=0, in_open:
;               Attach tap file with name at IX, drive in A
;       B=1, in_close:
;               Detach tap file
;       B=2, in_info:
;               Return attached filename to buffer at IX and drive in A
;       B=3, in_setpos:
;               Set position of tape pointer to block DE (0=start)
;       B=4, in_getpos:
;               Get position of tape pointer, in blocks, to HL
;       B=5, in_pause:
;               Toggles pause delay when loading SCREEN$
;               On exit, A=1 if pause now enabled, A=0 if not
;       B=6, in_flags:
;               Set tape flags to A
;               bit 0: 1=pause delay at SCREEN$ (as set by in_pause)
;               bit 1: 1=simulate tape loading with border/sound


; ************************************************************************
; * M_TAPEOUT ($8c)                                                      *
; ************************************************************************
; Tape output redirection control.
; Entry:
;       B=0, out_open:
;               Create/attach tap file with name at IX for appending, drive A
;       B=1, out_close:
;               Detach tap file
;       B=2, out_info:
;               Return attached filename to buffer at IX and drive in A
;       B=3, out_trunc:
;               Create/overwrite tap file with name at IX, drive A


; ************************************************************************
; * M_GETHANDLE ($8d)                                                    *
; ************************************************************************
; Get the file handle of the currently running dot command
; Entry:
;       -
; Exit:
;       A=handle
;       Fc=0
;
; NOTES:
; This call allows dot commands which are >8K to read further data direct
; from their own file (for loading into another memory area, or overlaying
; as required into the normal 8K dot command area currently in use).
; On entry to a dot command, the file is left open with the file pointer
; positioned directly after the first 8K.
; This call returns meaningless results if not called from a dot command.
```

```
; ************************************************************************
; * M_GETDATE ($8e)                                                      *
; ************************************************************************
; Get the current date/time.
; Entry:
;       -
; Exit:
;       Fc=0 if RTC present and providing valid date/time, and:
;               BC=date, in MS-DOS format
;               DE=time, in MS-DOS format
;       Fc=1 if no RTC, or invalid date/time, and:
;               BC=0
;               DE=0


; ************************************************************************
; * M_EXECCMD ($8f)                                                      *
; ************************************************************************
; Execute a dot command.
; Entry:
;       IX=address of commandline, excluding the leading "."
;          terminated with $00 (or $0d, or ':')
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code (0 means user-defined error)
;       HL=address of user-defined error message within dot command
;
; NOTES:
; The dot command name can be fully-pathed if desired. If just a name is
; provided, it is opened from the C:/BIN directory.
;   eg: defm    "hexdump afile.txt",0           ; runs c:/bin/hexdump
;       defm    "./mycommand.dot afile.txt",0   ; runs mycommand.dot in current
;                                               ; directory
; If A=0, the dot command has provided its own error message but this is not
; normally accessible. It can be read using the M_GETERR hook.
; This hook cannot be used from within another dot command.
```

```
; ******************************************************************************
; * M_DRVAPI ($92)                                                             *
; ******************************************************************************
; Access API for installable drivers.
; Entry:
;       C=driver id (0=driver API)
;       B=call id
;       HL,DE=other parameters
; Exit (success):
;       Fc=0
;       other values depend on API call
; Exit (failure):
;       Fc=1
;       A=0, driver not found
;       else A=driver-specific error code (esxDOS error code for driver API)

; If C=0, the driver API is selected and calls are as follows:
; (Note that these are not really useful for user applications; they are used
; by the .install/.uninstall dot commands).
;
; B=0, query the RTC
; (returns the same results as M_GETDATE)
;
; B=1, install a driver
;       D=number of relocations (0-255)
;       E=driver id, with bit 7=1 if should be called on an IM1 interrupt
;       HL=address of 512-byte driver code followed by D x 2-byte reloc offsets
; Possible error values are:
;       esx_eexist (18)         driver with same id already installed
;       esx_einuse (23)         no free driver slots available
;       esx_eloadingko (26)     bad relocation table
;
; B=2, uninstall a driver
;       E=driver id (bit 7 ignored)
;
; B=3, get paging value for driver banks
;       C=port (always $e3 on ZXNext)
;       A=paging value for DivMMC bank containing drivers (usually $82)
```

```
; ***************************************************************************
; * M_GETERR ($93)                                                          *
; ***************************************************************************
; Entry:
;       A=esxDOS error code, or 0=user defined error from dot command
;       if A=0, IX=error message address from dot command
;
;       B=0, generate BASIC error report (does not return)
;       B=1, return error message to 32-byte buffer at DE
;
; NOTES:
; Dot commands may use this call to fetch a standard esxDOS error message
; (with B=1), but must not use it to generate an error report (with B=0) as
; this would short-circuit the tidy-up code.
; User programs may use the call to generate any custom error message (and not
; just a custom message returned by a dot command). To do this, enter with
; A=0 and IX=address of custom message, where IX>=$4000.
; Custom error messages must be terminated with bit 7 set on the final
; character.


; ***************************************************************************
; * M_P3DOS ($94)                                                           *
; ***************************************************************************
; Make a +3DOS/IDEDOS/NextOS API call.
; Entry:
;       DE=+3DOS/IDEDOS/NextOS call ID
;       C=RAM bank that needs to be paged (usually 7, but 0 for some calls)
;       B'C',D'E',H'L',AF,IX contain entry parameters for call
; Exit:
;       exit values as described for +3DOS/IDEDOS/NextOS call ID
;       EXCEPT: any value to be returned in IX will instead be in H'L'
;       All registers except IX,IY may be changed.
;
; NOTES:
; Do not attempt to use this hook code unless you are running in NextOS mode
; (can be determined by using the M_DOSVERSION hook).
; Any parameters which are addresses of data (eg filenames etc) must lie between
; $4000...$BFE0.
; Any errors returned will be +3DOS/IDEDOS/NextOS error codes, not esxDOS error
; codes. Additionally, carry flag RESET indicates an error condition.
; No $DFFD paging should be in force.
; MMU2 ($4000-$5fff) must be the default (lower half of RAM bank 5), containing
; the system variables.
; The stack should be in normal configuration (not in TSTACK).
; For calls requiring normal configuration (ROM2/5/2/0), RAM0 must already
; be paged. For other calls, any banks can be paged at $c000, and will be
; restored when the +3DOS call has completed.
```

```
; ******************************************************************************
; * M_ERRH ($95)                                                               *
; ******************************************************************************
; Install error handler for dot command.
; Entry: HL=address of error handler within dot command
;           (0 to change back to standard handler)
;
; NOTES:
; Can only be used from within a dot command.
; If any BASIC error occurs during a call to ROM3 (using RST $10 or RST $18)
; then your error handler will be entered with:
;        DE=address that would have been returned to if the error had not
;           occurred
;        A=BASIC error code-1 (eg 8=9 STOP statement)
```

```
; ***************************************************************************
; * F_OPEN ($9a)                                                            *
; ***************************************************************************
; Open a file.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=filespec, null-terminated
;       B=access modes, a combination of:
;         any/all of:
;           esx_mode_read           $01         request read access
;           esx_mode_write          $02         request write access
;           esx_mode_use_header     $40         read/write +3DOS header
;         plus one of:
;           esx_mode_open_exist     $00         only open existing file
;           esx_mode_open_creat     $08         open existing or create file
;           esx_mode_creat_noexist  $04         create new file, error if exists
;           esx_mode_creat_trunc    $0c         create new file, delete existing
;
;       DE=8-byte buffer with/for +3DOS header data (if specified in mode)
;       (NB: filetype will be set to $ff if headerless file was opened)
; Exit (success):
;       Fc=0
;       A=file handle
; Exit (failure):
;       Fc=1
;       A=error code


; ***************************************************************************
; * F_CLOSE ($9b)                                                           *
; ***************************************************************************
; Close a file or directory.
; Entry:
;       A=file handle or directory handle
; Exit (success):
;       Fc=0
;       A=0
; Exit (failure):
;       Fc=1
;       A=error code


; ***************************************************************************
; * F_SYNC ($9c)                                                            *
; ***************************************************************************
; Sync file changes to disk.
; Entry:
;       A=file handle
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
```

```
; ***************************************************************************
; * F_READ ($9d)                                                            *
; ***************************************************************************
; Read bytes from file.
; Entry:
;       A=file handle
;       IX=address
;       BC=bytes to read
; Exit (success):
;       Fc=0
;       BC=bytes actually read (also in DE)
;       HL=address following bytes read
; Exit (failure):
;       Fc=1
;       BC=bytes actually read
;       A=error code
;
; NOTES:
; EOF is not an error, check BC to determine if all bytes requested were read.


; ***************************************************************************
; * F_WRITE ($9e)                                                           *
; ***************************************************************************
; Write bytes to file.
; Entry:
;       A=file handle
;       IX=address
;       BC=bytes to write
; Exit (success):
;       Fc=0
;       BC=bytes actually written
; Exit (failure):
;       Fc=1
;       BC=bytes actually written


; ***************************************************************************
; * F_SEEK ($9f)                                                            *
; ***************************************************************************
; Seek to position in file.
; Entry:
;       A=file handle
;       BCDE=bytes to seek
;       IXL=seek mode:
;                esx_seek_set      $00   set the fileposition to BCDE
;                esx_seek_fwd      $01   add BCDE to the fileposition
;                esx_seek_bwd      $02   subtract BCDE from the fileposition
; Exit (success):
;       Fc=0
;       BCDE=current position
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTES:
; Attempts to seek past beginning/end of file leave BCDE=position=0/filesize
; respectively, with no error.
```

```
; ***************************************************************************
; * F_FGETPOS ($a0)                                                         *
; ***************************************************************************
; Get current file position.
; Entry:
;       A=file handle
; Exit (success):
;       Fc=0
;       BCDE=current position
; Exit (failure):
;       Fc=1
;       A=error code


; ***************************************************************************
; * F_FSTAT ($a1)                                                           *
; ***************************************************************************
; Get file information/status.
; Entry:
;       A=file handle
;       IX=11-byte buffer address
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTES:
; The following details are returned in the 11-byte buffer:
;   +0(1)   '*'
;   +1(1)   $81
;   +2(1)   file attributes (MS-DOS format)
;   +3(2)   timestamp (MS-DOS format)
;   +5(2)   datestamp (MS-DOS format)
;   +7(4)   file size in bytes


; ***************************************************************************
; * F_FTRUNCATE ($a2)                                                       *
; ***************************************************************************
; Truncate/extend file.
; Entry:
;       A=file handle
;       BCDE=new filesize
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTES:
; Sets the filesize to precisely BCDE bytes.
; If BCDE<current filesize, the file is trunctated.
; If BCDE>current filesize, the file is extended. The extended part is erased
; with zeroes.
; The file position is unaffected. Therefore, if truncating, make sure to
; set the file position within the file before further writes (otherwise it
; will be extended again).
; +3DOS headers are included as part of the filesize. Truncating such files is
; not recommended.
```

```
; ***************************************************************************
; * F_OPENDIR ($a3)                                                         *
; ***************************************************************************
; Open directory.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=directory, null-terminated
;       B=access mode (only esx_mode_use_header and esx_mode_use_lfn matter)
;         any/all of:
;           esx_mode_use_lfn        $10         return long filenames
;           esx_mode_use_header     $40         read/write +3DOS headers
; Exit (success):
;       A=dir handle
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTES:
; Access modes determine how entries are formatted by F_READDIR.


; ***************************************************************************
; * F_READDIR ($a4)                                                         *
; ***************************************************************************
; Read next directory entry.
; Entry:
;       A=handle
;       IX=buffer
; Exit (success):
;       A=number of entries returned (0 or 1)
;         If 0, there are no more entries
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; Buffer format:
;  1 byte   file attributes (MSDOS format)
;  ? bytes  file/directory name, null-terminated
;  2 bytes  timestamp (MSDOS format)
;  2 bytes  datestamp (MSDOS format)
;  4 bytes  file size
;
; NOTES:
; If the directory was opened with the esx_mode_use_lfn bit, long filenames
; (up to 260 bytes plus terminator) are returned; otherwise short filenames
; (up to 12 bytes plus terminator) are returned.
; If opened with the esx_mode_use_header bit, after the normal entry follows the
; 8-byte +3DOS header (for headerless files, type=$ff, other bytes=zero).
```

```
; ******************************************************************************
; * F_TELLDIR ($a5)                                                            *
; ******************************************************************************
; Get current directory position.
; Entry:
;       A=handle
; Exit (success):
;       BCDE=current offset in directory
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


; ******************************************************************************
; * F_SEEKDIR ($a6)                                                            *
; ******************************************************************************
; Set current directory position.
; Entry:
;       A=handle
;       BCDE=offset in directory to seek to (as returned by F_TELLDIR)
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


; ******************************************************************************
; * F_REWINDDIR ($a7)                                                          *
; ******************************************************************************
; Rewind directory position to the start of the directory.
; Entry:
;       A=handle
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


; ******************************************************************************
; * F_GETCWD ($a8)                                                             *
; ******************************************************************************
; Get current working directory.
; Entry:
;       A=drive
;       IX=buffer for null-terminated path
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
```

```
; ******************************************************************************
; * F_CHDIR ($a9)                                                              *
; ******************************************************************************
; Change directory.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=path, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


; ******************************************************************************
; * F_MKDIR ($aa)                                                              *
; ******************************************************************************
; Create directory.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=path, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


; ******************************************************************************
; * F_RMDIR ($ab)                                                              *
; ******************************************************************************
; Remove directory.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=path, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
```

```
; ***************************************************************************
; * F_STAT ($ac)                                                            *
; ***************************************************************************
; Get unopened file information/status.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=filespec, null-terminated
;       DE=11-byte buffer address
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTES:
; The following details are returned in the 11-byte buffer:
;   +0(1)   drive specifier
;   +1(1)   $81
;   +2(1)   file attributes (MS-DOS format)
;   +3(2)   timestamp (MS-DOS format)
;   +5(2)   datestamp (MS-DOS format)
;   +7(4)   file size in bytes


; ***************************************************************************
; * F_UNLINK ($ad)                                                          *
; ***************************************************************************
; Delete file.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=filespec, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


; ***************************************************************************
; * F_TRUNCATE ($ae)                                                        *
; ***************************************************************************
; Truncate/extend unopened file.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=source filespec, null-terminated
;       BCDE=new filesize
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTES:
; Sets the filesize to precisely BCDE bytes.
; If BCDE<current filesize, the file is trunctated.
; If BCDE>current filesize, the file is extended. The extended part is erased
; with zeroes.
; +3DOS headers are included as part of the filesize. Truncating such files is
; not recommended.
```

```
; ***************************************************************************
; * F_CHMOD ($af)                                                           *
; ***************************************************************************
; Modify file attributes.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=filespec, null-terminated
;       B=attribute values bitmap
;       C=bitmap of attributes to change (1=change, 0=do not change)
;
;       Bitmasks for B and C are any combination of:
;           A_WRITE     %00000001
;           A_READ      %10000000
;           A_RDWR      %10000001
;           A_HIDDEN    %00000010
;           A_SYSTEM    %00000100
;           A_ARCH      %00100000
;
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


; ***************************************************************************
; * F_RENAME ($b0)                                                          *
; ***************************************************************************
; Rename or move a file.
; Entry:
;       A=drive specifier (overridden if filespec includes a drive)
;       IX=source filespec, null-terminated
;       DE=destination filespec, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


; ***************************************************************************
; * F_GETFREE ($b1)                                                         *
; ***************************************************************************
; Gets free space on drive.
; Entry:
;       A=drive specifier
; Exit (success):
;       Fc=0
;       BCDE=number of 512-byte blocks free on drive
; Exit (failure):
;       Fc=1
;       A=error code
```

Streaming API example - stream.asm

```
; ***************************************************************************
; * Streaming file access example code for NextOS via esxDOS API            *
; ***************************************************************************
; Assemble with: pasmo stream.asm stream.bin
;
; Execute with stream.bin and test.scr (any 6912-byte headerless screen file)
; in the same directory, using:
;
; CLEAR 32767:LOAD "stream.bin" CODE 32768
; LET x=USR 32768
;
; PRINT x to show any esxDOS error code on return.
; Additionally, 255 means "out of data"
; and 65535 means "completed successfully".


; ***************************************************************************
; * esxDOS API and other definitions required                               *
; ***************************************************************************

; Calls
f_open                  equ     $9a             ; opens a file
f_close                 equ     $9b             ; closes a file
disk_filemap            equ     $85             ; obtains map of file data
disk_strmstart          equ     $86             ; begin streaming operation
disk_strmend            equ     $87             ; end streaming operation

; File access modes
esx_mode_read           equ     $01             ; read access
esx_mode_open_exist     equ     $00             ; open existing files only

; Next registers
next_register_select    equ     $243b
nxr_peripheral2         equ     $06

; Size of filemap buffer (in 6-byte entries)
; To guarantee all entries will fit in the filemap at once, allow 1 entry for
; every 2K of filesize. The example uses a 6.75K SCREEN$, so 4 entries is
; sufficient.
; (NOTE: Reducing this to 1 *may* force the example code to refill the filemap
;        multiple times, but only if your card has a cluster size of 2K or 4K
;        and the file is fragmented).
filemap_size            equ     4


; ***************************************************************************
; * Initialisation                                                          *
; ***************************************************************************

        org     $8000

; Before starting we will disable the Multiface button, since filesystem
; access will not be possible during a streaming operation, and could cause
; unexpected effects, including possibly the machine locking up until a soft
; reset is performed.

        ld      bc,next_register_select
        ld      a,nxr_peripheral2
        out     (c),a
```

```
        inc     b
        in      a,(c)                   ; get current peripheral2 value
        and     %11110111               ; clear bit 3 (multiface enable)
        out     (c),a

; First the file must be opened in the normal way

        ld      a,'*'                   ; use default drive if none specified
        ld      ix,test_filename
        ld      b,esx_mode_read+esx_mode_open_exist
        rst     $08
        defb    f_open
        jp      c,exit_with_error
        ld      (filehandle),a          ; store the returned file handle

; For this example, we are going to "stream" a standard Spectrum SCREEN$
; file to the screen. This is a convenient point to set up parameters
; for this.

        ld      hl,$4000                ; address to stream data to
        ld      de,6912                 ; size of data left to stream
        exx                             ; save in alternate registers


; ***************************************************************************
; * Filemap buffer setup                                                    *
; ***************************************************************************

; Next, obtain the map of card addresses for the file.
; Note that this call (DISK_FILEMAP) must be made directly after opening the
; file - no other file access calls should be made first.
;
; A buffer must be provided to hold the card addresses.
;
; Each entry in the buffer occupies 6 bytes and describes an area of the
; file which can be anywhere between 2K and 32MB in size (depending on the
; way the card was formatted, and how fragmented the file is).
; Therefore, it is possible to calculate the absolute maximum number of buffer
; entries required by dividing the size of the file by 2K.
;
; It is also possible to use a smaller buffer and call disk_filemap multiple
; times when a refill is required (provided the last streaming operation has
; been stopped before the next disk_filemap call is made).
;
; Often, files are unfragmented, and so will use only 1 entry. You could
; potentially write your code to assume this (which would therefore be simpler
; than this example), and cause an error if more than 1 entry is returned,
; citing "framentation" and suggesting the user run the .defrag dot command
; on the file. (Note that some CompactFlash, and other IDE, may be limited
; to a maximum section size of 64K).
;
; The byte/block addressing flag returned in bit 1 of A may be useful if you
; wish to start streaming data from a particular 512-byte block offset within
; the file, as it indicates how to adjust the 4-byte card addresses:
;   if bit 1 of A=0, then add 512 to the card address for every block
;   if bit 1 of A=1, then add 1 to the card address for every block

refill_map:
        ld      a,(filehandle)
        ld      ix,filemap_buffer       ; address of buffer
        ld      de,filemap_size         ; size of buffer (in 6-byte entries)
        rst     $08
```

```
        defb    disk_filemap
        jp      c,close_and_exit_with_error

; On exit from disk_filemap, the return values are:
;       DE=size of buffer unused (in 6-byte entries)
;       HL=address in buffer after last written entry
;       A=flags: bit 0=card id (0 or 1)
;                bit 1=0 for byte addressing, 1 for block addressing

        ld      (cardflags),a           ; store card flags for later use

; First we will check whether there were any entries returned, and exit with
; a dummy error code ($ff) not used by esxDOS to indicate "out of data" if not.

        push    hl
        ld      de,filemap_buffer       ; initialise buffer address
;       and     a                       ; not needed as no error, so carry=0
        sbc     hl,de                   ; any entries in the buffer at all?
        pop     hl
        ld      a,$ff                   ; dummy error to indicate out of data
        jr      z,close_and_exit_with_error


; ***************************************************************************
; * Main streaming loop                                                     *
; ***************************************************************************
; Now we can enter a loop to stream data from each entry in the buffer.

stream_loop:
        push    hl                      ; save buffer end address
        ex      de,hl                   ; HL=address of next entry in buffer
        ld      e,(hl)
        inc     hl
        ld      d,(hl)
        inc     hl
        ld      c,(hl)
        inc     hl
        ld      b,(hl)                  ; BCDE=card address
        inc     hl
        push    bc
        pop     ix                      ; IXDE=card address
        ld      c,(hl)
        inc     hl
        ld      b,(hl)                  ; BC=number of 512-byte blocks
        inc     hl
        push    hl                      ; save updated buffer address
        push    bc                      ; save number of blocks

; Streaming is initiated by calling DISK_STRMSTART with:
;       IXDE=card address
;       BC=number of 512-byte blocks to stream
;       A=card flags, as returned by DISK_FILEMAP
; After this call is issued it is important that no further esxDOS calls
; (or NextOS calls which might access a filesystem) are issued until the
; matching DISK_STRMEND call has been made.
; It is also important to ensure that the Multiface (which could access files)
; is disabled for the duration of the streaming operation. (Done earlier in
; this example).

        ld      a,(cardflags)           ; A=card flags
        rst     $8
        defb    disk_strmstart
```

```
        pop     ix                      ; retrieve number of blocks to IX
        jr      c,drop2_close_and_exit_with_error

; If successful, the call returns with:
;       B=protocol: 0=SD/MMC, 1=IDE
;       C=data port
; NOTE: On the Next, these values will always be:
;       B=0
;       C=$EB
; Therefore, your code code be slightly faster and simpler if writing a
; Next-only program. However, these values are provided to allow portable
; streaming code to be written (if NextOS is later ported to other platforms).

        ld      a,c
        exx                             ; switch back to "streaming set"
                                        ; HL=address, DE=bytes to stream
        ld      c,a                     ; C=data port


; ***************************************************************************
; * Block streaming loop                                                    *
; ***************************************************************************

stream_block_loop:
        ld      b,0                     ; prepare for 256-byte INIR

        ld      a,d
        cp      2                       ; at least 1 block to stream?
        jr      c,stream_partial_block

; Read an entire 512-byte block of data.
; These could be unrolled to INIs for maximum performance.

        inir                            ; read 512 bytes from the port
        inir
        dec     d                       ; update byte count
        dec     d

; Check the protocol being used.
        exx
        ld      a,b                     ; A=protocol (0=SD/MMC, 1=IDE)
        exx
        and     a                       ; The IDE protocol doesn't need
        jr      nz,protocol_ide         ; this end-of-block processing

; For SD protocol we must next skip the 2-byte CRC for the block just read.
; Note that maximum performance of the interface is 16T per byte, so nops
; must be added if not using INI/OUTI.
; The interface can run at CPU speeds of at least 21MHz (as in ZX-Badaloc).

        in      a,(c)
        nop
        in      a,(c)
        nop

; And then wait for a token of $FE, signifying the start of the next block.
; A value of $FF indicates "token not yet available". Any other value is an
; error.

wait_token:
        in      a,(c)                   ; wait for start of next block
        cp      $ff                     ; (a token is != $ff)
```

```
        jr      z,wait_token
        cp      $fe                     ; the correct data token is $fe
        jr      nz,token_error          ; anything else is an error

; IDE protocol streaming can rejoin here.
protocol_ide:
        ld      a,d                     ; check if any more bytes needed
        or      e
        jr      z,streaming_complete

        dec     ix                      ; decrement block count
        ld      a,ixl
        or      ixh
        jr      nz,stream_block_loop    ; continue until all blocks streamed

        exx                             ; switch "streaming set" to alternates


; ************************************************************************
; * Main streaming loop end                                              *
; ************************************************************************
; After all the 512-byte blocks for a particular card address have been
; streamed, the DISK_STRMEND call must be made. This just requires A=cardflags.

        ld      a,(cardflags)
        rst     $08
        defb    disk_strmend
        jr      c,drop2_close_and_exit_with_error

; Following disk_strmend, the system is back in a state where any other esxdos
; calls may now be used, including (if necessary) DISK_FILEMAP to refill the
; buffer. This can be an expensive call, though, so it would be preferable to
; ensure that the buffer is large enough to be filled with the first call.
; This would also simplify the code a little.

        pop     de                      ; DE=current buffer address
        pop     hl                      ; HL=ending buffer address
;       and     a                       ; not needed; carry=0 since no error
        sbc     hl,de                   ; any more entries left in buffer?
        jr      z,refill_map            ; if not, refill
        add     hl,de                   ; re-form ending address
        jr      stream_loop             ; back for next entry in the buffer


; ************************************************************************
; * Stream a partial block                                               *
; ************************************************************************
; It is entirely okay to stream a partial block, since the streaming operation
; can be terminated at any point by issuing the DISK_STRMEND call.

stream_partial_block:
        and     a                       ; at least 256 bytes left?
        jr      z,stream_final_bytes

        inir                            ; read 256 bytes from the port

stream_final_bytes:
        ld      b,e
        inc     b
        dec     b
        jr      z,streaming_complete
```

```
        inir                            ; read last few bytes from the port

streaming_complete:
        ld      a,(cardflags)
        rst     $08
        defb    disk_strmend            ; terminate the streaming operation
        jr      drop2_close_and_exit_with_error


; ***************************************************************************
; * Tidy up and exit                                                        *
; ***************************************************************************

token_error:
        ld      a,$ff                   ; dummy error to indicate out of data
        scf

drop2_close_and_exit_with_error:
        pop     hl                      ; discard buffer addresses
        pop     hl

close_and_exit_with_error:
        push    af                      ; save error status

        ld      a,(filehandle)
        rst     $08
        defb    f_close

        pop     af                      ; restore error status

exit_with_error:
        ld      hl,$2758
        exx                             ; BASIC requires H'L'=$2758 on return
        ld      b,0
        ld      c,a                     ; BC=error, for return to BASIC
        ret     c                       ; exit if there was an error
        ld      bc,$ffff                ; use 65535 to indicate "no error"
        ret


; ***************************************************************************
; * Data                                                                    *
; ***************************************************************************

test_filename:
        defm    "test.scr",0            ; filenames must be null-terminated

filehandle:
        defb    0

filemap_buffer:
        defs    filemap_size*6          ; allocate 6 bytes per entry

cardflags:
        defb    0
```

Driver example (file 1 of 2) - border.asm

```
; ***************************************************************************
; * Simple example NextOS driver                                            *
; ***************************************************************************
;
; This file is the 512-byte NextOS driver itself, plus relocation table.
;
; Assemble with: pasmo border.asm border.bin border.sym
;
; After this, border_drv.asm needs to be built to generate the actual
; driver file.


; ***************************************************************************
; * Entry points                                                            *
; ***************************************************************************
; Drivers are a fixed length of 512 bytes (although can have external 8K
; banks allocated to them if required).
;
; They are always assembled at origin $0000 and relocated at installation time.
;
; Your driver always runs with interrupts disabled, and may use any of the
; standard register set (AF,BC,DE,HL). Index registers and alternates must be
; preserved.
;
; No esxDOS hooks or restarts may be used. However, 3 calls are provided
; which drivers may use:
;
;       call    $2000   ; drv_drvswapmmc
;                       ; Used for switching between allocated DivMMC banks
;
;       call    $2003   ; drv_drvrtc
;                       ; Query the RTC. Returns BC=date, DE=time (as M_DATE)
;
;       call    $2006   ; drv_drvapi
;                       ; Access other drivers. Same parameters as M_DRVAPI.
;
; The stack is always located below $4000, so if ZX banks have been allocated
; they may be paged in at any location (MMU2..MMU7). However, when switching
; to other allocated DivMMC banks, the stack cannot be used unless you set
; it up/restore it yourself.
; If you do switch any banks, don't forget to restore the previous MMU settings
; afterwards.


; ***************************************************************************
; * Entry points                                                            *
; ***************************************************************************

        org     $0000

; At $0000 is the entry point for API calls directed to your driver.
; B,DE,HL are available as entry parameters.

; If your driver does not provide any API, just exit with A=0 and carry set.
; eg:
;       xor     a
;       scf
;       ret
```

```
api_entry:
        jr      border_api
        nop

; At $0003 is the entry point for the interrupt handler. This will only be
; called if bit 7 of the driver id byte has been set in your .DRV file, so
; need not be implemented otherwise.

im1_entry:
reloc_1:
        ld      a,(colour)
        inc     a                       ; increment stored border colour
        and     $07
reloc_2:
        ld      (colour),a
        out     ($fe),a                 ; set it
        ret


; ************************************************************************
; * Simple example API                                                   *
; ************************************************************************
; On entry, use B=call id with HL,DE other parameters.
; (NOTE: HL will contain the value that was either provided in HL (when called
;        from dot commands) or IX (when called from a standard program).
;
; When called from the DRIVER command, DE is the first input and HL is the
second.
;
; When returning values, the DRIVER command will place the contents of BC into
; the first return variable, then DE and then HL.

border_api:
        bit     7,b                     ; check if B>=$80
        jr      nz,channel_api          ; on if so, for standard channel API

        djnz    bnot1                   ; On if B<>1

; B=1: set values.

reloc_3:
        ld      (value1),de
reloc_4:
        ld      (value2),hl
        and     a                       ; clear carry to indicate success
        ret

; B=2: get values.

bnot1:
        djnz    bnot2                   ; On if B<>2
reloc_5:
        ld      a,(colour)
        ld      b,0
        ld      c,a
reloc_6:
        ld      de,(value1)
reloc_7:
        ld      hl,(value2)
        and     a                       ; clear carry to indicate success
        ret
```

```
; Unsupported values of B.

bnot2:
api_error:
        xor     a                       ; A=0, unsupported call id
        scf                             ; Fc=1, signals error
        ret


; ***************************************************************************
; * Standard channel API                                                    *
; ***************************************************************************
; If you want your device driver to support standard channels for i/o, you
; can do so using the following API calls.
; Each call is optional - just return with carry set and A=0
; for any calls that you don't want to provide.
;
; B=$f9: open channel
; B=$fa: close channel
; B=$fb: output character
; B=$fc: input character
; B=$fd: get current stream pointer
; B=$fe: set current stream pointer
; B=$ff: get stream size/extent

channel_api:
        ld      a,b
        sub     $f9                     ; set zero flag if call $f9 (open)
        jr      c,api_error             ; exit if invalid ($80..$f8)
        ld      b,a                     ; B=0..6
        jr      nz,bnotf9               ; on if not $f9 (open)


; B=$f9: open channel
; In the documentation for your driver you should describe how it should be
; opened. The command used will determine the input parameters provided to
; this call (this example assumes your driver id is ASCII 'X', ie $58):
; OPEN #n,"D>X"         ; simple open: HL=DE=0
; OPEN #n,"D>X>string"  ; open with string: HL=address, DE=length
;                       ; NOTE: be sure to check for zero-length strings
; OPEN #n,"D>X,p1,p2"   ; open with numbers: DE=p1, HL=p2 (zeros if not
provided)
;
; This call must return a unique channel handle in A. This allows your driver
; to support multiple different concurrent channels if desired.
;
; If you return with any error (carry set), "Invalid filename" will be reported
; and no stream will be opened.
;
; For this example, we will simply check that no other channels have yet been
; opened:

reloc_8:
        ld      a,(chanopen_flag)
        and     a
        jr      nz,api_error            ; exit with error if already open
        ld      a,1
reloc_9:
        ld      (chanopen_flag),a       ; signal "channel open"
        ret                             ; exit with carry reset (from AND above)
                                        ; and A=handle=1
```

```
; B=$fa: close channel
; This call is entered with D=handle, and should close the channel
; If it cannot be closed for some reason, exit with an error (this will be
; reported as "In use").

bnotf9:
        djnz    bnotfa                  ; on if not call $fa
reloc_10:
        call    validate_handle         ; check D is our handle (does not return
                                        ; if invalid)
        xor     a
reloc_11:
        ld      (chanopen_flag),a       ; signal "channel closed"
        ret                             ; exit with carry reset (from XOR)


; B=$fb: output character
; This call is entered with D=handle and E=character.
; If you return with carry set and A=$fe, the error "End of file" will be
; reported. If you return with carry set and A<$fe, the error
; "Invalid I/O device" will be reported.
; Do not return with A=$ff and carry set; this will be treated as a successful
; call.

bnotfa:
        djnz    bnotfb                  ; on if not call $fb
reloc_12:
        call    validate_handle         ; check D is our handle (does not return
                                        ; if invalid)
reloc_13:
        ld      a,(output_ptr)
reloc_14:
        call    calc_buffer_add         ; HL=address within buffer
        ld      (hl),e                  ; store character
        inc     a
        and     $1f
reloc_15:
        ld      (output_ptr),a          ; update pointer
        ret                             ; exit with carry reset (from AND)


; B=$fc: input character
; This call is entered with D=handle.
; You should return the character in A (with carry reset).
; If no character is currently available, return with A=$ff and carry set.
; This will cause INPUT # or NEXT # to continue calling until a character
; is available.
; If you return with carry set and A=$fe, the error "End of file" will be
; reported. If you return with carry set and any other value of A, the error
; "Invalid I/O device" will be reported.

bnotfb:
        djnz    bnotfc                  ; on if not call $fc
reloc_16:
        call    validate_handle         ; check D is our handle (does not return
                                        ; if invalid)
reloc_17:
        ld      a,(input_ptr)
reloc_18:
        call    calc_buffer_add         ; HL=address within buffer
        ld      e,(hl)                  ; get character
```

```
        inc     a
        and     $1f
reloc_19:
        ld      (input_ptr),a           ; update pointer
        ld      a,e                     ; A=character
        ret                             ; exit with carry reset (from AND)


; B=$fd: get current stream pointer
; This call is entered with D=handle.
; You should return the pointer in DEHL (with carry reset).

bnotfc:
        djnz    bnotfd                  ; on if not call $fd
reloc_20:
        call    validate_handle         ; check D is our handle (does not return
                                        ; if invalid)
reloc_21:
        ld      a,(input_ptr)
        ld      l,a
        ld      h,0                     ; HL=stream pointer
        ld      d,h
        ld      e,h
        and     a                       ; reset carry (successful call)
        ret


; B=$fe: set current stream pointer
; This call is entered with D=handle and IXHL=pointer.
; Exit with A=$fe and carry set if the pointer is invalid (will result in
; an "end of file" error).
; NOTE: Normally you should not use IX as an input parameter, as it cannot
;       be set differently to HL if calling via the esxDOS-compatible API.
;       This call is a special case that is only made by NextOS.

bnotfd:
        djnz    bnotfe                  ; on if not call $fe
reloc_22:
        call    validate_handle         ; check D is our handle (does not return
                                        ; if invalid)
        ld      a,l                     ; check if pointer >$1f
        and     $e0
        or      h
        or      ixl
        or      ixh
        scf
        ld      a,$fe
        ret     nz                      ; exit with A=$fe and carry set if so
        ld      a,l
reloc_23:
        ld      (input_ptr),a           ; set the pointer
        and     a                       ; reset carry (successful call)
        ret


; B=$ff: get stream size/extent
; This call is entered with D=handle
; You should return the size/extent in DEHL (with carry reset).

bnotfe:
reloc_24:
        call    validate_handle         ; check D is our handle (does not return
```

```
                                        ; if invalid)
        ld      hl,32                   ; our simple channel is always size 32
        ld      d,h
        ld      e,h
        and     a                       ; reset carry (successful call)
        ret


; ************************************************************************
; * Validate handle for our simple channel                               *
; ************************************************************************

validate_handle:
        dec     d                       ; D should have been 1
        ret     z                       ; return if so
        pop     af                      ; otherwise discard return address
        jr      api_error               ; and exit with error


; ************************************************************************
; * Validate handle for our simple channel                               *
; ************************************************************************

calc_buffer_add:
        push    af                      ; save offset into buffer
reloc_25:
        ld      hl,channel_data         ; base address
        add     a,l                     ; add on offset
        ld      l,a
        ld      a,0
        adc     a,h
        ld      h,a
        pop     af                      ; restore offset
        ret


; ************************************************************************
; * Data                                                                 *
; ************************************************************************

colour:
        defb    0

value1:
        defw    0

value2:
        defw    0

chanopen_flag:
        defb    0

input_ptr:
        defb    0

output_ptr:
        defb    0

channel_data:
        defs    32
```

```
; ***************************************************************************
; * Relocation table                                                        *
; ***************************************************************************
; This follows directly after the full 512 bytes of the driver.

        defs    512-$

if ($ != 512)
.ERROR Driver code exceeds 512 bytes
endif

; Each relocation is the offset of the high byte of an address to be relocated.

reloc_start:
        defw    reloc_1+2
        defw    reloc_2+2
        defw    reloc_3+3
        defw    reloc_4+2
        defw    reloc_5+2
        defw    reloc_6+3
        defw    reloc_7+2
        defw    reloc_8+2
        defw    reloc_9+2
        defw    reloc_10+2
        defw    reloc_11+2
        defw    reloc_12+2
        defw    reloc_13+2
        defw    reloc_14+2
        defw    reloc_15+2
        defw    reloc_16+2
        defw    reloc_17+2
        defw    reloc_18+2
        defw    reloc_19+2
        defw    reloc_20+2
        defw    reloc_21+2
        defw    reloc_22+2
        defw    reloc_23+2
        defw    reloc_24+2
        defw    reloc_25+2
reloc_end:
```

Driver example (file 2 of 2) - border_drv.asm

```
; ***************************************************************************
; * Simple example NextOS driver file                                       *
; ***************************************************************************
;
; This file generates the actual border.drv file which can be installed or
; uninstalled using the .install/.uninstall commands.
;
; The driver itself (border.asm) must first be built.
;
; Assemble this file with: pasmo border_drv.asm border.drv


; ***************************************************************************
; * Definitions                                                             *
; ***************************************************************************
; Pull in the symbol file for the driver itself and calculate the number of
; relocations used.

        include "border.sym"

relocs  equ     (reloc_end-reloc_start)/2


; ***************************************************************************
; * .DRV file header                                                        *
; ***************************************************************************
; The driver id must be unique, so current documentation on other drivers
; should be sought before deciding upon an id. This example uses $7f as a
; fairly meaningless value. A network driver might want to identify as 'N'
; for example.

        org     $0000

        defm    "NDRV"          ; .DRV file signature

        defb    $7f+$80         ; 7-bit unique driver id in bits 0..6
                                ; bit 7=1 if to be called on IM1 interrupts

        defb    relocs          ; number of relocation entries (0..255)

        defb    0               ; number of additional 8K DivMMC RAM banks
                                ; required (0..8)

        defb    0               ; number of additional 8K Spectrum RAM banks
                                ; required (0..200)


; ***************************************************************************
; * Driver binary                                                           *
; ***************************************************************************
; The driver + relocation table should now be included.

        incbin  "border.bin"


; ***************************************************************************
; * Additional bank images and patches                                      *
; ***************************************************************************
; If any 8K DivMMC RAM banks or 8K Spectrum RAM banks were requested, then
```

```
; preloaded images and patch lists should be provided.
;
;       First, for each mmcbank requested:
;
;       defb    bnk_patches     ; number of driver patches for this bank id
;       defw    bnk_size        ; size of data to pre-load into bank (0..8191)
;       defs    bnk_size        ; data to pre-load into bank
;       defs    bnk_patches*2   ; for each patch, a 2-byte offset (0..511) in
;                               ; the 512-byte driver to write the bank id to
;       NOTE: The first patch for each mmcbank should never be changed, as
;             .uninstall will use the value for deallocating.
;
;       Then, for each zxbank requested:
;
;       defb    bnk_patches     ; number of driver patches for this bank id
;       defw    bnk_size        ; size of data to pre-load into bank (0..8191)
;       defs    bnk_size        ; data to pre-load into bank
;       defs    bnk_patches*2   ; for each patch, a 2-byte offset (0..511) in
;                               ; the 512-byte driver to write the bank id to
;       NOTE: The first patch for each zxbank should never be changed, as
;             .uninstall will use the value for deallocating.
```

List of updates

Updates: 28 Jan 2018
Added new M_DRVAPI hook providing acceess to a new API for installable drivers.

Added new IDE_DRIVER call to access new driver API from +3DOS.

Added notes on the new driver API and optional driver channel API, with a worked example (border.asm & border_drv.asm).

Rewrote the notes about dot commands.

Added **RST $20** facility to terminate a dot command and bootstrap a game/application.

Updates: 18 Jan 2018
Added more information about dot commands.

Added **M_GETHANDLE**, **M_EXECCMD** and **M_GETERR** hooks.

Updates: 17 Jan 2018
Added note about turning off layer 2 writes.

Added note about layer 2 banks in IDE_BANK call.

Updates: 15 Jan 2018
Added general descriptions of the **+3DOS**-compatible and **esxDOS**-compatible APIs.

Added full documentation for the **esxDOS**-compatible API.

Updates: 12 Dec 2017
Updated details of the **IDE_GET_LFN** call. This now additionally returns the file's size and last update time & date.

Added new **IDE_RTC** call for querying the real-time-clock (if present).

Updates: 30 Nov 2017
Updated details of the **IDE_BROWSER** call. This now has a capabilities mask allowing selected functionality to be enabled or disabled as desired. Also added note about using as a save file dialog.

Updates: 23 Nov 2017
The **IDE_STREAM_LINEIN** call has been removed and replaced by a new **IDE_WINDOW_LINEIN** call.

Added new **IDE_INTEGER_VAR** call for accessing NextBASIC integer variables.

Noted that the **IDE_STREAM_*** calls may corrupt the alternate register set, in addition to the effects on the standard register set noted for each individual call. (The special note about memory configuration has also been removed for the **IDE_WINDOW_*** calls; this applies only to the **IDE_STREAM_*** calls).

Updates: 14 Nov 2017
Added note that it is now possible to use the wildcard character ***** in the

IDE_BROWSER call to match remaining characters in the filetype (with examples).

Added more notes on the IDE_STREAM_LINEIN call.

Added new IDE_WINDOW_STRING call.