NextZXOS and esxDOS APIs (Updated 24 May 2023)

This document describes the **NextZXOS API** (as at NextZXOS v2.08), 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
        NextZXOS Editor features

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

Available APIs

NextZXOS provides 2 distinct and separate APIs:
- a **+3DOS**-compatible API, providing the main **NextZXOS** 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 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 a set of general file-access calls, which are implemented as thin layers on top of a lower level filesystem-access API (not currently exposed to the user). The **esxDOS**-compatible API is generally easier to use, but 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.

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).

Some calls (eg IDE_BROWSER) may access the system variables region (eg for keyboard scanning or other purposes). Therefore, you should generally ensure that IY still points to the system variable ERR_NR ($5c3a) before making such a call.

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 **NextZXOS**):

```
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
%IDE_MOUNT ($01d2)           Unmount/remount SD cards
```

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
%IDE_MODE ($01d5)                Query NextBASIC display mode info, or change mode
%IDE_TOKENISER ($01d8)           Convert BASIC between plain text & tokenised forms
```

The following API calls are related to floppy drives and will not be useful for most software (included for use of legacy software running from .DSK images):

```
DOS_REF_XDPB ($0151)             Point at XDPB for low level disk access
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_ASK_1 ($017B)                 Is unit 1 (external drive) present?
DD_DRIVE_STATUS ($017E)          Fetch drive status
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_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_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
DOS_MAP_B ($0154)                Map B: onto unit 0 or 1
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_EQUIPMENT ($0181)          What type of drive?
DD_L_WRITE ($0193)            uPD765A write driver


IDE_FORMAT ($00B2)            Format a partition
IDE_PARTITION_NEW ($00B8)     Create partition
IDE_PARTITION_INIT ($00BB)    Initialise 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.

As well as the changes described here, the following calls take a 16K page number in either C or B which indicates what memory should be present at $c000..$ffff for the read/write operation. On the +3/+3e, page numbers 0-7 were allowed; on *NextZXOS* any valid 16K RAM page 0-111 may be used:

DOS_READ (0112h)
DOS_WRITE (0115h)
IDE_SWAP_OUT (00DFh)
IDE_SWAP_IN (00E2h)

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, shared-write or shared-read-write
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
                        6 = shared-write
                        7 = shared-read-write
                Bit 3: 0=open file, 1=open associated metadata file
                Bits 4...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 type 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
                bit 3 = 0 (reserved)

```
                    bit 4 = suppress new sort/filter even when preloaded entry is
                            zeroised
                    bit 5 = enable sort/filter mode in A
                    bit 6 = 0 (reserved)
                    bit 7 = use 2nd catalog handle
              A=sort/filter mode, if bit 5 of C set:
                    bit 7=exclude files
                    bit 6=exclude dirs
                    bit 5=exclude special . and .. entries
                    bit 4=exclude system/hidden files
                    bit 3=enable sorting
                    bit 2=reverse sort mode
                    bits 1..0=sort mode:
                                $00=LFNs
                                $01=short 8.3 name
                                $02=date/time (LFN breaks ties)
                                $03=file size (LFN breaks ties)
              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).
                C = 0 if sorting operation not completed
                      (eg out of memory, user pressed BREAK)
                C <> 0 if sorting completed
                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
```

NOTES:
Sorting/filtering and the 2nd catalog handle are only supported from NextZXOS
v2.07 onwards.

The filter modes in A (if specified) override the legacy filter bits in C. If
bit 5 of C is not set (ie filter modes in A not specified), the default
behaviour is to sort on short (8.3) filenames.

Sorting uses memory allocated by IDE_BANK. The memory will be freed on the next
unsorted call to DOS_CATALOG. To do this, it is advised to use B=2, C=$20 (or
$a0 if using the 2nd catalog handle), A=0 and your original DE/HL parameters.

Sorting/filtering only occurs when the preloaded entry is zeroed. You can
additionally prevent a re-sort/filter operation by setting bit 4 of C. This can
be useful if you want to show/process the first set of results again without
needing another (potentially slow) sorting operation.

Two independent catalog handles are available, selected by bit 7 of C. This
allows two independently sorted and filtered buffers of catalog data to be used
simultaneously. The Browser uses this feature in "unmixed" mode (handle 0 is
used for an unfiltered list of directories, and handle 1 is used for a filtered
list of files).

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_SWAP_OPEN ($00D9)

Open a swap file

IN: A(bits 6..0)=block size in sectors, 1 (0.5K) to 32 (16K)
     If bit 7 of A is 0, then:
         BC=max block number required
         and NextZXOS will open an available system swap file
         (c:/nextzxos/swp-N.p3s) large enough
     If bit 7 of A is 1, then:
         BC=$ff-terminated name of file to use
         (maximum block number will be determined from file)

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

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

NOTE: The block size specified (any multiple of 0.5K up to 16K) determines the
amount of data that is swapped in and out with the other IDE_SWAP_ calls. The
size of the swap partition required is calculated as (blocksize)* (max block
number+1). The current block number is set to 0.

NOTE: Only unfragmented files can be opened as swap files. The error code
rc_fragmented ($4a) will be returned for fragmented files.

NOTE: From NextZXOS v2.07, any unfragmented file (not just swap files created by
.mkswap) can be specified as a swap file. This makes it easier to use generated
resource files through the swap APIs.

IDE_SWAP_EX ($00E5)

IN: -
OUT(f): Fc=0, A=rc_notimp

NOTE: This call is inefficient and has been deprecated.
      Use **IDE_SWAP_IN** and **IDE_SWAP_OUT** instead.

IDE_DOS_MAP ($00F1)

Map a drive to the specified partition or physical device

```
IN:  A=unit (0..15), including special device:
                  4=RAMdisk
                  $ff=filesystem image (.P3D or .DSK file)
     BC=partition number (if A!=4 or $ff)
     BC=image filename ($ff-terminated) (if A=$ff)
     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_DOS_MAPPING ($00F7)

Obtain mapping information for the specified drive

```
IN:  L=drive letter 'A' to 'P' (uppercase)
     BC=buffer (18 bytes in length)
        (on exit, will be filled with description, padded with spaces)

or (from NextZXOS v2.07):
     L=$80 + drive letter (uppercase)
     BC=buffer (256 bytes in length)
        (on exit, will be filled with description, terminated with $ff)

OUT(s):  Fc=1
         Fz=1 if drive not mapped (and other info not valid)
         Fz=0, mapping is as follows:
           A=unit (0..15), including special device:
                        4=RAMdisk
                        $ff=filesystem image (.P3D or .DSK file)
           BC=partition number (not for special devices)

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 type .Z80, .SNA, .SNX, .O
and .P are supported, with others potentially supported in future).

.SNX files are loaded in the same way as .SNA files, with the following
exceptions:
   1. The machine is left in Next mode with all hardware features still enabled,
      rather than being locked into 48K or 128K mode.
   2. File handle 0 is left open to the .SNX file. This allows additional data
      to be loaded from the file, by using the esxDOS F_SEEK hook to set the
      filepointer to the location of additional data and the F_READ hook to read
      it (+3DOS calls typically cannot be used as the snapshot loading process
      overwrites +3DOS data held in RAM page 7).
   3. .SNX files of 64K size or greater are always treated as 128K snapshots.

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).

Note that for rc_path_change, the current default drive is *not* changed; only the current directory for the specified drive. To change the default drive, use the DOS_SET_DRIVE call (and, optionally, change the system variables LODDRV and/or SAVDRV which affect the default drives for NextBASIC's LOAD/SAVE/VERIFY/MERGE commands).

New calls

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

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 type (letters must be capitalised),
                        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
           $20, BROWSERCAPS_UNMOUNT- drives may be unmounted
           $40, BROWSERCAPS_CAPS2  - further capabilities specified in B
           $80, BROWSERCAPS_SYSCFG - system use only - use browser.cfg
     Alternatively just use one of the two special values:
           $00, BROWSERCAPS_NONE   - no special capabilities
           $3f, BROWSERCAPS_ALL    - all capabilities enabled
     If BROWSERCAPS_CAPS2 is set, extended capabilities are present in B:
           $01, BROWSERCAPS2_RUNEXEC- .RUN directories may be executed
           $02, BROWSERCAPS2_GUIDE  - guides may be shown (does not return)
           $04, BROWSERCAPS2_EXTEND - extended menu may be shown (does not
                                                                 return)
           $08, BROWSERCAPS2_SORT   - directory sorting allowed (uses IDE_BANK)

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
                  DE=address of long filename (terminated with $ff) in RAM 7
         If Fz=0, SPACE/BREAK was pressed
OUT(f):  Fc=0, A=error
```

From NextZXOS v2.07, if DE=0, the Browser preferences are accessed:

```
IN:  DE=0, get/set Browser settings
     HL=15-byte buffer for $ff-terminated filter,
        or 96-byte buffer for colour schemes
     C=current view flags:
           bits 1..0: 00=no extra info
                      01=file sizes
                      10=date/time
                      11=attributes
           bit 2: reserved (do not change)
           bit 3: reserved (do not change)
           bit 4: 1=show system files
           bit 5: 1=hide "." and ".."
           bit 6: 1=show dirs before files (dirs will not be filtered)
           bit 7: reserved (do not change)
     B=current sort flags:
           bits 1..0: 00=LFNs
                      01=unsorted
                      10=date/time
                      11=file size
           bit 2: reverse sorting
           bits 3..7: reserved (do not change)
     A=0, get current settings to C, B, and buffer at HL
     A=1, change settings to C, B and filespec at HL
```

```
        A=2, get current Browser colour scheme to buffer at HL
        A=3, change Browser colour scheme to buffer at HL
        A=4, set browser palette & ATTR_P
        A=5, restore previous palette & ATTR_P
        A=6, get current Editor colour scheme to buffer at HL
        A=7, change Editor colour scheme to buffer at HL
```

OUT: Fc=1, success

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

NOTES:

BROWSERCAPS_CAPS2 is only supported from NextZXOS v2.07.

IY must point to the system variable ERR_NR ($5c3a) on entry to this call.

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)
        rc_bank_available (4), return number of currently-available 8K banks
                               of specified type
     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
         E=available number of 8K banks of specified type, for rc_bank_available

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.

Banks are allocated starting with the highest-numbered available bank. This helps to ensure low-numbered banks remain available for longer (such banks are often allocated explicitly in NextBASIC programs).

NextZXOS/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.

NextZXOS/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).

If you intend to return to BASIC, don't forget to first clear the ERR_NR system
variable back to $ff (no error).

Additionally, note that a m/code program using IDE_BASIC should not be executed
using a variable assignment, eg:
     LET variable=USR x
since this will cause unexpected effects if the BASIC statements executed by
IDE_BASIC have also performed any variable assignments.

This will not work for adding lines to a program (ie no line number should be
present).

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.

      From NextZXOS v2.07 onwards:
        If A=0, then:
          C=buffer size (1..255)
          B=flags: bit 0: exit after any change (delete/insert char)
                   bit 1: exit if invalid key pressed
                   bit 2: D contains starting cursor position
                   bit 4: rewind print position (by E chars) before starting
                   bit 5: allow token/graphics entry
                   bit 6: enable BREAK to generate "H STOP in INPUT" error
                          (inhibited if %CODE bit 1 set)
                   bit 7: use current mode's full-screen window rather
                          than currently-set user window channel
          D=starting cursor position (0..E), only if B bit 2 is set

OUT: E=number of characters returned in input buffer

      From NextZXOS v2.07 onwards:
        B=flags: bit 0: exit was due to delete/insert char, not ENTER
                 bit 1: exit was due to invalid key
        D=final cursor position
        C=invalid key (if bit 1 set in B)

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, DELETE and EDIT.

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).

Editing can be suspended if desired when the input data has changed or an
unknown key has been pressed, allowing your code to perform appropriate actions
before optionally continuing the edit by re-calling IDE_WINDOW_LINEIN.

On exit, the print position follows the input data regardless of the reason for
exiting. To continue editing the same data, either restore the print position
before calling IDE_WINDOW_LINEIN again or (from NextZXOS v2.08) set bit 4 of the
flags in B.

From NextZXOS v2.07, setting bit 7 of B allows the current mode's full-screen
window to be used, rather than a user-defined window channel. Layer 0 does not
have a full-screen window. In this case, input will be performed in either the
main screen area or lower-screen area, depending upon whether the current
channel is "S" (usually stream 2) or "K" (usually stream 0 or 1).

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
         H=secs to 1-second precision (time in DE only provides 2-sec precision)
         L=100ths of second (or $ff if not supported by RTC module)

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 NextZXOS 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).

IDE_MOUNT ($01d2)

Unmount/remount SD cards

```
IN:      A=0, close all files, unmap all drives and swap partitions
         A=1, mount SD cards and automap drives

OUT(s):  Fc=1

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

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

NOTE:

This call can be used to allow users to change SD cards, as if the REMOUNT
command was being executed.
First, call IDE_MOUNT with A=0 to close all files and unmap drives.
If successful, request the user to change the SD card(s) and then call IDE_MOUNT
with A=1 to mount the new SD cards and automap drives.

IDE_MODE ($01d5)

Query current NextBASIC display mode information, or change mode

IN: A=0, query current mode information
         A=1, change mode to:
             B=layer (0,1,2)
             C=sub-mode (if B=1): 0=lo-res, 1=ula, 2=hi-res, 3=hi-col

OUT(f): Fc=0
         A=rc_badparam (bad parameter)

OUT(s): Fc=1

         A=current (or new) mode/layer (same as lower 4 bits of GMODE):
             bits 0..1=layer (0,1,2)
             bits 2..3=sub-mode for layer 1 (0=lores,1=ula,2=hires,3=hicol)

         H=printable lines on screen:
             22 for layer 0
             12 for lo-res, standard height printing
             16 for lo-res, reduced height printing
             24 for ula/hi-res/hi-col/layer2, standard height printing
             32 for ula/hi-res/hi-col/layer2, reduced height printing

         L=printable columns on screen:
             32 for layer 0
             from 16 to 170 on other modes, depending on character size

         E=current attributes, for layer 0/ula/hi-res/hi-col
           current ink, for lo-res/layer2

         D=current paper, for lo-res/layer2

         B=character width in pixels (3-8)

         C=flags: bit 0=1 if reduced-height mode is currently in force
                  bit 4=1 if double-width mode is currently in force
                  bit 5=1 if double-height mode is currently in force

         IX=mode window handle (not valid if A=0, layer 0 - unchanged from entry
            value)

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

NOTE: H and L don't take account of the double-width/height flags in C, so
      if those bits are set then the current number of printable lines/columns
      will be half the reported values. (The number of lines in H *does* take
      account of the reduced height setting, bit 0 of C).

NOTE: For layer 1 and 2 modes, the mode window handle is returned in IX. This
      can be stored in the system variable CURCHL before making calls to
      IDE_WINDOW_STRING or IDE_WINDOW_LINEIN, so that these calls use the
      full-screen mode windows. It is important to restore the original value
      of CURCHL after doing this.
      For layer 0, IX is unchanged. However, IDE_WINDOW_LINEIN can still be used
      by setting CURCHL to 0. As IDE_MODE doesn't change IX if the current mode
      is layer 0, so you can set IX=0 before calling IDE_MODE and use the
      returned value of IX with IDE_WINDOW_LINEIN regardless of mode.

NOTE: Changing the mode does *not* cause the screen to be cleared. This can be
      useful if switching between layer 0 and layer 1,1 (or if switching between
      layer 2 and one of the other modes, since layer 2 uses different memory
      to the ULA modes).

Simple dot commands can just use the standard RST $10 call to output characters
and assume a screen width of 32 characters, which will work regardless of the
current layer/mode.

The information provided by this call can be useful if, however, you want to
write a dot command that respects the user's current display settings, and
formats output appropriately to use the entire screen.

If the current mode is layer 0 (ie A=0 on return from this call), you can clear
the screen using a standard 48K ROM call:
      rst $18
      defw $0D6B ; 48K ROM CLS call

For all other layers/modes this will not work correctly. Instead you should just
send the "clear window" control code using RST $10:
      ld   a,14 ; clear window control code
      rst $10

For all layers/modes except for layer 0, you can also use the other window
control codes in this way (for example to change character width, enable double-
width/height etc). It is good practice to restore any settings that you change
before exiting your dot command.

If you wish to use the windowing controls but the current mode is layer 0, you
can use this call to first change to layer 1 mode 1. However, be sure to change
the mode back to layer 0 before exiting the dot command.

IDE_TOKENISER ($01d8)
Available from NextZXOS v2.01

Convert BASIC between plain text & tokenised forms

IN: B=0, tokenise BASIC line (plain text -> tokenised)
          B=1, detokenise BASIC line (tokenised -> plain text)
          C=8K bank containing buffer for untokenised BASIC line (ASCII text)
          HL=offset in bank of buffer for untokenised BASIC line ($0000..$1fff)

          If tokenising (B=0):
             the untokenised BASIC line should be terminated with ENTER ($0d)

          If detokenising (B=1):
             DE=address of tokenised BASIC line within normal BASIC workspace,
                terminated with ENTER ($0d)
             The recommended way to reserve and use this space is:
                (i) Call the 48K ROM routine SET_WORK ($16bf) to clear the
                      workspace area
                (ii) Call the 48K ROM routine BC_SPACES ($0030) to reserve
                       enough room in the workspace area to hold the tokenised
                       line (enter with BC=line length, including the ENTER)
                (iii) Copy the tokenised line (including the ENTER) to the
                        address returned by BC_SPACES in the DE register, and
                        use this as the DE entry parameter for IDE_TOKENISER
             Alternatively, you could map a bank containing the entire
             tokenised line into any of MMU3/4/5 and provide an address to
             there. Care must be taken that there is no possibility of the
             stack being in the MMU that you have selected.

OUT(f): Fc=0
         A=rc_badparam (bad parameter)

OUT(s): Fc=1

         If tokenising (B=0):
            HL=(E_LINE)=address of (partly or fully) tokenised line
            BC=length of tokenised line, including ENTER ($0d)
            Fz=0, line successfully tokenised and syntax-checked
            Fz=1, syntax error occurred, and:
                   DE=count of successfully processed chars in untokenised line

         If detokenising (B=1):
            HL=offset in provided buffer following the detokenised line

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

NOTES:

Only attempt to use this call if running on at least v2.01 of NextZXOS.

This call can be useful in preparing a BASIC line to be executed by IDE_BASIC.

Line numbers should NOT be present at the start of the input buffer.

If tokenising, the tokenised line should be reclaimed after use by calling
RECLAIM_2 ($19e8) in ROM3, with the values of HL & BC returned by this call.
This effectively restores the original contents of the E_LINE area (which
directly follow the newly-tokenised line) and allows a return to BASIC to be
made successfully.

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
74    rc_fragmented     File is fragmented, use .DEFRAG
```

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:/DOT/ directory, but they can be run from anywhere if fully-pathed. For example:

```
.mydot          ; executes C:/DOT/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 type.

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 routine with **RST $10** or **RST $18,** or the **M_P3DOS** hook code for +3DOS API calls using the ROM2/5/2/0 configuration).

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). A ':' character within double-quotes does *not* indicate the end of the command line. For example the termination of the following command line is the second ':', not the first:
```
     .mydot "c:/dir/file":
```

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

NextZXOS 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 its id does not clash with any other existing driver. However, it is 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 all use the 'P' identifier). Further information can be found on the current Next SD distribution in the file c:/docs/nextzxos/DriverIDs.txt.

Keyboard driver
In addition to the 4 general-purpose drivers, it is also possible to replace the standard keyboard driver with a 512-byte driver. This is defined in the same way, except that it always has a fixed id (0) and provides only a single entry point, for the interrupt routine; no driver API is supported for this special driver.

It might be desired to replace the standard keyboard driver in order to support different international keyboard layouts, or perhaps to add support for a multi-keystroke buffer.

An example keyboard driver (keyboard.asm and keyboard_drv.asm) is available separately, and included at the end of this document.

Printer drivers
The id "P" is reserved for printer drivers. If such a driver is installed in the system then NextBASIC will automatically send any output on #3 (ie LLIST, LPRINT, PRINT #3 etc) to it. CP/M will also use any such driver as its LPT device.

CP/M will also use any driver with id "X" as its AUX device. AUX drivers can be written in a similar way to printer drivers.

An example printer driver (sample_prt.asm and sample_prt_drv.asm) is available separately, and included at the end of this document.

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*
GOTO *#n,value*   (set current stream pointer)
RETURN *#n* **TO** *var* (get current stream pointer to variable *var*)
DIM *#n* **TO** *var*   (get current stream size/extent to variable *var*)
NEXT *#n* **TO** *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 NextZXOS 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_autoload         ; $90 (144)            load a BASIC program from tape or disk
m_setcaps          ; $91 (145)            set additional capabilities
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/NextZXOS 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 [HL from dot command]=buffer (must be located >= $4000)
;       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.
;
; If the file has been accessed, the filepointer should be reset to the start
; using F_SEEK, and a single byte read (with F_READ) before making this call.
; This will ensure that the current sector information maintained by the OS
; is correctly pointing to the first sector of the file.
;
; The provided buffer address must be >=$4000 (ie dot commands will need to
; allocate space in the main RAM area using the BC_SPACES call in the 48K
; ROM, or page in an allocated bank).
;
; If you wish to check whether a file is unfragmented, there are 2 ways:
;   (1) for files < ~32MB in size, make a call to DISK_FILEMAP. If there is
;       only 1 entry (ie exit HL=entry HL+6), the file is unfragmented
;   (2) for files > ~32MB in size, you must manually check whether each
;       section of the file directly follows the previous one. The .DEFRAG
;       dot command contains appropriate code for this that you may wish to
;       use: please see the source in src/asm/dot_commands/defrag.asm on the
;       standard SD card distribution.
;
; Please see example application code, stream.asm, for full usage information
; (available separately or at the end of this document).
```

```
; ****************************************************************************
; * DISK_STRMSTART ($86)                                                     *
; ****************************************************************************
; Start reading from the card in streaming mode.
; Entry: IXDE [HLDE from dot command]=card address
;        BC=number of 512-byte blocks to stream [not used for SD/MMC protocol]
;        A=card flags
;        additionally, from NextZXOS v2.01, bit 7 may be set to indicate that
;        the user will perform the initial wait for data token
; Exit (success): Fc=0
;                 B=0 for SD/MMC protocol, 1 for IDE protocol
;                 C=8-bit data port
; Exit (failure): Fc=1, A=esx_edevicebusy
;
; NOTES:
;
; On the Next, this call always returns with B=0 (SD/MMC protocol) and C=$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.
;
; If the protocol is known in advance to be SD/MMC (ie on the Next), the value
; of BC is ignored. Streaming will always continue indefinitely, until a
; DISK_STRMEND call is issued.
;
; On NextZXOS v2.01+, you may optionally set bit 7 of A to indicate that the
; call should return without waiting for the initial $FE data token, allowing
; other work to be done to cover the latency. In this case, the user must wait
; for the $FE token before any data is read from the stream.
;
; 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 NextZXOS:
;                 Fc=0, success
;                 B='N',C='X' (NextZXOS signature)
;                 DE=NextZXOS version in BCD format: D=major, E=minor version
;                    eg for NextZXOS v1.94, DE=$0194
;                 HL=language code:
;                       English: L='e',H='n'
;                       Spanish: L='e',H='s'
;                       Further languages may be available in the future
;                 A=0 if running in NextZXOS mode (and zero flag is set)
;                 A<>0 if running in 48K mode (and zero flag is reset)


; ***************************************************************************
; * M_GETSETDRV ($89)                                                       *
; ***************************************************************************
; Get or set the default drive.
; Entry:
;       A=0, get the default drive
;       A<>0, set the default drive to A
;             bits 7..3=drive letter (0=A...15=P)
;             bits 2..0=ignored (use 1 to ensure A<>0)
; Exit (success):
;       Fc=0
;       A=default drive, encoded as:
;             bits 7..3=drive letter (0=A...15=P)
;             bits 2..0=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTE:
; This call isn't often useful, as it is not necessary to provide a
; specific drive to calls which need a drive/filename.
; For such calls, you can instead provide:
;   A='*'   use the default drive
;   A='$'   use the system drive (C:, where the NEXTZXOS and BIN dirs are)
; Any drive provided in such calls is also overridden by any drive letter
; that is specified in the filename (eg "D:/myfile.txt\0").
;
; NOTE:
; When setting a drive, this call only affects the default drive seen by other
; esxDOS and NextZXOS API calls. It does *not* change the default LOAD/SAVE
; drives used by NextBASIC, which are stored in the LODDRV and SAVDRV system
; variables.
```

```
; ****************************************************************************
; * M_TAPEIN ($8b)                                                           *
; ****************************************************************************
; Tape input redirection control.
; Entry:
;       B=0, in_open:
;               Attach tap file with name at IX [HL from dot command],
;               drive in A
;       B=1, in_close:
;               Detach tap file
;       B=2, in_info:
;               Return attached filename to buffer at IX [HL from dot command],
;               and drive (always '*' from NextZXOS v2.07) in A
;       B=3, in_setpos:
;               Set position of tape pointer to block DE (0=start)
;       B=4, in_getpos:
;               Get position of tape pointer, in blocks, to HL
;       B=5, in_pause:
;               Toggles pause delay when loading SCREEN$
;               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
;               On exit, A=previous value of the tape flags


; ****************************************************************************
; * M_TAPEOUT ($8c)                                                          *
; ****************************************************************************
;
; Tape output redirection control.
; Entry:
;       B=0, out_open:
;               Create/attach tap file with name at IX [HL from dot command]
;               for appending, drive A
;       B=1, out_close:
;               Detach tap file
;       B=2, out_info:
;               Return attached filename to buffer at IX [HL from dot command]
;               and drive (always '*' from NextZXOS v2.07) in A
;       B=3, out_trunc:
;               Create/overwrite tap file with name at IX [HL from dot command],
;               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
;               H=secs to 1-second precision
;                 (time in DE only provides 2-sec precision)
;               L=100ths of second (or $ff if not supported by RTC module)
;       Fc=1 if no RTC, or invalid date/time, and:
;               BC=0
;               DE=0
;               HL undefined


; ****************************************************************************
; * M_EXECCMD ($8f)                                                          *
; ****************************************************************************
; Execute a dot command.
; Entry:
;       IX [HL from dot command]=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:/DOT directory.
;   eg: defm    "hexdump afile.txt",0           ; runs c:/dot/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_AUTOLOAD ($90)                                                         *
; ****************************************************************************
; Load a BASIC program from tape or disk.
; Entry: A=0:
;               load next BASIC program from tape (or .TAP, if redirected)
;         or
;         A=drive specifier
;         IX [HL from dot command]=filename
;               load named BASIC program from disk
;
; Exit (success):
;       does not return if successfull
; Exit (failure):
;       Fc=1
;       A=error code
```

```
; *************************************************************************
; * M_SETCAPS ($91)                                                       *
; *************************************************************************
; Entry: A=capabilities to set:
;             bit 7=1, do not erase new file data in f_truncate/f_ftruncate
;                      (increases performance of these calls)
;             bits 0..6: reserved, must be zero
; Exit:  Fc=0, success
;        E=previous capabilities
;
; NOTE: This call is only available from NextZXOS v1.98M+.
;       Earlier versions will return with Fc=1 (error) and A=esx_enocmd
; NOTE: You should save the original value of the capabilities which is
;       returned in E. After completing the calls you need with your altered
;       capabilities, restore the original value by calling M_SETCAPS again
;       with the value that was previously returned in E.
;       This will ensure that other programs running after you have exited
;       will continue to see the original expected behaviour.
```

```
; **************************************************************************
; * 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
;
; B=4, get driver image
;       E=driver id (bit 7 ignored)
;       HL=address of 512-byte buffer

; NOTES:
; Any provided buffer addresses must be located >=$4000 since the lower 16K of
; memory is occupied with driver memory when this call is in operation.
```

```
; ****************************************************************************
; * M_GETERR ($93)                                                           *
; ****************************************************************************
; Entry:
;       A=esxDOS error code, or 0=user defined error from dot command
;       if A=0, IX [HL from dot command]=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 [HL from dot command]=address of custom message, >=$4000.
; Custom error messages must be terminated with bit 7 set on the final
; character.


; ****************************************************************************
; * M_P3DOS ($94)                                                            *
; ****************************************************************************
; Make a +3DOS/IDEDOS/NextZXOS API call.
; Entry:
;       DE=+3DOS/IDEDOS/NextZXOS 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 [HL from dot command] contain entry parameters
;       for call
; Exit:
;       exit values as described for +3DOS/IDEDOS/NextZXOS 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:
;
; B'C', D'E', H'L' contain the entry parameters that the +3DOS API call
; expects to be in BC, DE, HL.
;
; As with other esxDOS API calls, any IX entry parameter should instead be
; loaded into HL if making the call from within a dot command.
;
; Do not attempt to use this hook code unless you are running in NextZXOS 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/NextZXOS 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 with
; C=7 (ie requiring RAM7 at the top and the stack below $bfe0), M_P3DOS will
; automatically switch the stack into TSTACK during the call, so there is no
; need for calling code to adjust stack location before invoking M_P3DOS.
;
```

```
; For calls requiring normal configuration (ROM2/5/2/0), RAM0 must already
; be paged, and SP should still be in the region it was on entry to the dot
; command (between (STKEND) and (RAMTOP)).
; For other calls, any banks can be paged at $c000, and SP may be anywhere
; except within TSTACK.
```

```
; **************************************************************************
; * 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 ($9a)                                                           *
; **************************************************************************
; Open a file.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;         (overridden if filespec includes a drive)
;       IX [HL from dot command]=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 [HL from dot command]=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 [HL from dot command]=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 [L from dot command]=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 [HL from dot command]=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:
; The M_SETCAPS ($91) hook can be used to modify the behaviour of this call
; so that is doesn't zeroise additional file sections (improving performance).
; 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 ($a3)                                                          *
; ****************************************************************************
; Open directory.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;         (overridden if filespec includes a drive)
;       IX [HL from dot command]=directory, null-terminated
;
;       B=access mode - add together any or all of:
;       $00 esx_mode_short_only    F_READDIR returns short 8.3 names
;       $10 esx_mode_lfn_only      F_READDIR returns LFNs only
;       $18 esx_mode_lfn_and_short F_READDIR returns LFN followed by 8.3 name
;                                  (both null-terminated)
;       $20 esx_mode_use_wildcards F_READDIR only returns entries matching
;                                  wildcard
;       $40 esx_mode_use_header    F_READDIR additionally returns +3DOS header
;       $80 esx_mode_sf_enable     enable sort/filter mode in C
;
;       C=sort/filter mode (if enabled in access mode) – add together:
;       $80 esx_sf_exclude_files   F_READDIR doesn't return files
;       $40 esx_sf_exclude_dirs    F_READDIR doesn't return directories
;       $20 esx_sf_exclude_dots    F_READDIR doesn't return . or .. directories
;       $10 esx_sf_exclude_sys     F_READDIR doesn't return system/hidden files
;       $08 esx_sf_sort_enable     entries will be sorted
;                                  (unless memory exhausted/BREAK pressed)
;       $00 esx_sf_sort_lfn        sort by LFN
;       $01 esx_sf_sort_short      sort by short name
;       $02 esx_sf_sort_date       sort by date/time (LFN breaks ties)
;       $03 esx_sf_sort_size       sort by file size (LFN breaks ties)
;       $04 esx_sf_sort_reverse    sort order is reversed
;
;       DE=null-terminated wildcard string, if esx_mode_use_wildcards
;          The same string must also be passed when calling F_READDIR, in case
;          sorting is not possible and a fall back to unsorted mode is made.
; Exit (success):
;       A=dir handle
;       C=0 if sort operation not completed (out of memory/user pressed BREAK)
;       C<>0 if sorting completed
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTES:
; Sorting/filtering and esx_mode_lfn_and_short are only available from
; NextZXOS v2.07 onwards.
; Access modes determine how entries are formatted by F_READDIR.
```

```
; ****************************************************************************
; * F_READDIR ($a4)                                                          *
; ****************************************************************************
;
; Read next directory entry.
; Entry:
;       A=handle
;       IX [HL from dot command]=buffer
;       Additionally, if directory was opened with esx_mode_use_wildcards:
;       DE=wildcard string (null-terminated)
; Exit (success):
;       A=number of entries returned (0 or 1)
;          If 0, there are no more entries
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; Buffer format:
;  1 byte   file attributes (MSDOS format)
;  ? bytes  file/directory name(s), null-terminated
;  2 bytes  timestamp (MSDOS format)
;  2 bytes  datestamp (MSDOS format)
;  4 bytes  file size
;
; NOTES:
;
; If the directory was opened with esx_mode_lfn_only, long filenames
; (up to 260 bytes plus terminator) are returned.
; If the directory was opened with esx_mode_short_only, short filenames
; (up to 12 bytes plus terminator) are returned.
; If the directory was opened with esx_mode_lfn_and_short, long filenames
; and short filenames are both returned (LFN comes first).
;
; If opened with esx_mode_use_header, after the normal entry follows the
; 8-byte +3DOS header (for headerless files, type=$ff, other bytes=zero).
;
; If opened with esx_mode_use_wildcards, then only filenames
; matching the wildcard string provided in DE are returned.


; ****************************************************************************
; * 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 ($a8)                                                          *
; ***************************************************************************
;
; Get current working directory (or working directory for any filespec)
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;   or: A=$ff, to obtain working directory for a supplied filespec in DE
;       DE=filespec (only if A=$ff)
;       IX [HL from dot command]=buffer for null-terminated path
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
;
; NOTE:
; If obtaining a path for a supplied filespec, the filename part (after the
; final /, \ or :) is ignored so need not be provided, or can be the name of a
; non-existent file/dir.
;
;
; NOTE:
; If you want the current working directory for a drive other than the current
; or system drive, specify A=$ff and have DE point to a null-terminated
; drive specifier string (eg "E:"). Alternatively you can place an esxDOS drive
; specifier in A (as described in M_GETSETDRV).
;
;
; NOTE:
; IX [HL from dot command] and DE may both address the same memory, if desired.
```

```
; ***************************************************************************
; * F_CHDIR ($a9)                                                           *
; ***************************************************************************
; Change directory.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;          (overridden if filespec includes a drive)
;       IX [HL from dot command]=path, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTE: This hook changes the directory for the drive specified in A (or in
;       the path) but does not change the current drive. If this is required
;       you must also use the M_GETSETDRV hook.


; ***************************************************************************
; * F_MKDIR ($aa)                                                           *
; ***************************************************************************
; Create directory.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;          (overridden if filespec includes a drive)
;       IX [HL from dot command]=path, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code


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

```
; *************************************************************************
; * F_STAT ($ac)                                                          *
; *************************************************************************
; Get unopened file information/status.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;         (overridden if filespec includes a drive)
;       IX [HL from dot command]=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)   '*'
;   +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 ($ad)                                                        *
; *************************************************************************
; Delete file.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;         (overridden if filespec includes a drive)
;       IX [HL from dot command]=filespec, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTE: This call only deletes the base file, not any associated metadata file.
;       Use the +3DOS call DOS_DELETE to ensure any associated metadata file is
;       also deleted.
```

```
; ***************************************************************************
; * F_TRUNCATE ($ae)                                                        *
; ***************************************************************************
;
; Truncate/extend unopened file.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;         (overridden if filespec includes a drive)
;       IX [HL from dot command]=source filespec, null-terminated
;       BCDE=new filesize
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTES:
; The M_SETCAPS ($91) hook can be used to modify the behaviour of this call
; so that is doesn't zeroise additional file sections (improving performance).
; 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 ($af)                                                           *
; ***************************************************************************
;
; Modify file attributes.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;         (overridden if filespec includes a drive)
;       IX [HL from dot command]=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 ($b0)                                                          *
; ***************************************************************************
;
; Rename or move a file.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
;         (overridden if filespec includes a drive)
;       IX [HL from dot command]=source filespec, null-terminated
;       DE=destination filespec, null-terminated
; Exit (success):
;       Fc=0
; Exit (failure):
;       Fc=1
;       A=error code
;
; NOTE: This call only renames the base file, not any associated metadata file.
;       Use the +3DOS call DOS_RENAME to ensure any associated metadata file is
;       also renamed.


; ***************************************************************************
; * F_GETFREE ($b1)                                                         *
; ***************************************************************************
;
; Gets free space on drive.
; Entry:
;       A=drive specifier ('*'=default, '$'=system)
; 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 NextZXOS 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) should be made directly after opening the
; file - no other file access calls should be made first.
; (If the file has been accessed, the filepointer should be reset to the start
; using F_SEEK, and a single byte read (with F_READ) before making this call.
; This will ensure that the current sector information maintained by the OS
; is correctly pointing to the first sector of the file.)
;
; 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
;           (NOTE: not actually used for the SD/MMC protocol, so you can supply
;                  any value; streaming continues until DISK_STRMEND is used).
;       A=card flags, as returned by DISK_FILEMAP
; After this call is issued it is important that no further esxDOS calls
; (or NextZXOS 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).
; NOTE: From NextZXOS v2.01, you can optionally set bit 7 of A to indicate
;       that DISK_STRMSTART should not wait for the data from the card to
;       be ready to stream. This allows the user to do some other work first
;       to cover the latency and later manually check that the data is ready.
;       If you set this bit (and if the protocol is SD/MMC), before reading
;       any data you must first wait for the data token ($fe) to be received:
;       the required code is the same as at the later "wait_token" routine.

        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 NextZXOS 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.
```

```
            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 NextZXOS driver                                          *
; ****************************************************************************
;
;
; This file is the 512-byte NextZXOS 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:
;
;
;          jp       $2000    ; drv_drvswapmmc
;                            ; Can be used to aid switching between allocated
;                            ; DivMMC banks (see example usage below).
;
;
;          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).
;
;
; If using other allocated DivMMC banks, note that the stack location is the
; 224 bytes $260d..$26ec inclusive. Therefore, if you wish to switch to other
; DivMMC banks (in particular using the mechanism below) you should leave this
; region of memory unused in each of your allocated DivMMC banks (or avoid any
; use of the stack, or take care of switching SP whenever you switch banks).
;
;
; If you do switch any banks, don't forget to restore the previous MMU settings
; afterwards.
;
;
;
; ****************************************************************************
; * Switching between allocated DivMMC banks                                *
; ****************************************************************************
;
; You can request DivMMC banks to be allocated to your driver, as well as
; (or instead of standard ZX memory banks). However, DivMMC banks are a more
; limited resource and are more awkward to use, since they can only be paged
; in at $2000..$3fff (where your driver code is already running in another
; DivMMC bank).
;
;
; If you wish to use DivMMC banks, the following helper code is provided
```

```
; in the driver's DivMMC bank at $2000 (drv_drvswapmmc):
;       $2000:  out     ($e3),a
;               ret
;
; One suggested method for switching between your allocated DivMMC banks
; and your driver is as follows:
;
; 1. In the preload data for each DivMMC bank (specified in the .DRV
;    file), include the following routine at the start (ie $2000):
;       $2000:  out     ($e3),a
;               push    bc              ; save B=driver bank
;               jp      (hl)
;
; 2. Provide the following subroutine somewhere within your 512-byte driver
code:
;       call_externmmc:
;               push    af
;               in      a,($e3)
;               ld      b,a             ; save driver bank in B
;               pop     af
;               set     7,a             ; set bit 7 on DivMMC bank id to page
;               jp      $2000           ; jump to switch banks and "return"
;                                       ; to routine HL in external DivMMC bank
;
; 3. To call a routine in one of your allocated DivMMC banks, use this in
;    your driver code:
;               ld      hl,routineaddr
;               ld      a,divmmcbankid  ; (to be patched by .INSTALL)
;               call    call_externmmc
;
; 4. The routines in your allocated DivMMC banks should end with:
;               pop     af              ; A=driver bank id
;               jp      $2000           ; switch back to driver and return
;
; Don't forget that the stack takes up the region $260d..$26ec and so you
; should not use this region for any other purpose in your DivMMC banks if
; you are using this mechanism.


; **************************************************************************
; * 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,standard_api         ; on for standard API functions if so

        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 API functions                                                 *
; ***************************************************************************
;
; API calls $80..$ff are used in a standard way by NextZXOS.
;
;
; If (and only if) you have set bit 7 of the "mmcbanks" value in your
; driver file's header, then 2 special calls are made to allow you to
; perform any necessary initialisation or shutdown of your driver
; when it is .INSTALLed and .UNINSTALLed:
;
; B=$80: initialise
; B=$81: shutdown
;
; Each of these calls is made with the following parameters:
;   HL=address of structure containing:
;        byte 0: # of 8K ZX RAM banks allocated (as specified in .DRV header)
;        bytes 1+: list of bank ids for the allocated 8K ZX RAM banks
;   DE=address of structure containing:
;        byte 0: # of 8K DivMMC RAM banks allocated (as specified in .DRV header)
;        bytes 1+: list of bank ids for the allocated 8K DivMMC RAM banks
;
; These bank lists are in main RAM ($4000-$ffff) so be careful not to
; page them out during use. They are temporary structures and only
; available during the initialise ($80) and shutdown ($81) calls.
;
; Note that the initialise ($80) call is made after the allocated RAM
; banks have been erased and preloaded with data from your .DRV file.
; Most drivers will therefore probably not need to use these lists, as
; the allocated bank ids can also be patched directly into your driver
; code during the .INSTALL process.
;
; The shutdown ($81) call does NOT need to deallocate the RAM banks -
; this will be done by the .UNINSTALL dot command.
;
; When exiting the calls, return with carry clear to indicate success.
; If carry is set on call $80, the .INSTALL procedure will be aborted.
; If carry is set on call $81, the .UNINSTALL procedure will be aborted.

standard_api:
        ; The example border driver sets bit 7 of mmcbanks,
        ; so needs to provide API calls $80 and $81.
        ld      a,b
        and     $7f
        jr      z,driver_init           ; on for call $80, initialise
        dec     a
        jr      nz,channel_api          ; if not $81, must be a channel API call

; B=$81: shutdown driver
;        This call is optional and should be provided if you set bit 7 of
;        the mmcbanks value in the driver header.
;        Exit with carry clear if the driver can be safely UNINSTALLed, or
;        carry set to abort the UNINSTALL process.

driver_shutdown:
        and     a                       ; always safe to uninstall this driver
        ret

; B=$80: initialise driver
;        This call is optional and should be provided if you set bit 7 of
;        the mmcbanks value in the driver header.
;        Exit with carry clear if the driver can be safely INSTALLed, or
```

```
;          carry set to abort the INSTALL process.
;          This call is provided for drivers that might need additional
;          hardware initialisation.

driver_init:
          and     a                       ; always safe to install this driver
          ret

; The following calls are used to allow your driver to support
; channels for i/o (manipulated with BASIC commands like OPEN #).
; Each call is optional - just return with carry set and A=0
; for any calls that you don't want to provide.
;
; B=$f6: copy screen (usually for printer drivers)
; B=$f7: return output status
; B=$f8: return input status
; B=$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     $f7                     ; set zero flag if call $f7
                                          ; (return output status)
          jr      c,api_error             ; exit if unsupported (<$f7)
          ld      b,a                     ; B=0..8
          jr      nz,bnotf7               ; on if not $f7 (output status)


; B=$f7: return output status
; This call is entered with D=handle.
; You should return BC=$ffff if the device is ready to accept a character
; to be output, or BC=$0000 if it is not ready.
; NOTE: NextBASIC does not use this call for standard channel i/o, but it
;       may be useful to provide it for use by machine-code programs or
;       for NextBASIC programs using the DRIVER command.
; This call is also used by CP/M for printer drivers (with id "P") and
; AUX drivers (with id "X").

          ld      bc,$ffff                ; our device always ready for output
          and     a                       ; clear carry to indicate success
          ret


; B=$f8: return input status
; This call is entered with D=handle.
; You should return BC=$ffff if the device has an input character available
; to be read, or BC=$0000 if there is no character currently available.
; NOTE: NextBASIC does not use this call for standard channel i/o, but it
;       may be useful to provide it for use by machine-code programs or
;       for NextBASIC programs using the DRIVER command.
; This call is also used by CP/M for AUX drivers (with id "X").

bnotf7:
          djnz    bnotf8
          ld      bc,$ffff                ; our device always ready for input
          and     a                       ; clear carry to indicate success
          ret
```

```
; 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 should return a channel handle in A. This allows your driver
; to support multiple different concurrent channels if desired.
; If your device is simple you may choose to ignore the channel handles
; in this and other calls.
;
;
; If you return with any error (carry set), "Invalid filename" will be reported
; and no stream will be opened.
;
;
; For this example, we will only allow a single channel to be opened at
; a time, by performing a simple check:

bnotf8:
        djnz    bnotf9
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


; Subroutine to 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


; 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 NextZXOS.

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                                 *
; ****************************************************************************
;

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

; Our driver header will specify these values to be patched with the ids
; of the external banks allocated to us.
bankid_mmc0:
        defb    0
bankid_zx0:
        defb    0
bankid_zx1:
        defb    0
bankid_zx2:
        defb    0


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

if ($ > 512)
.ERROR Driver code exceeds 512 bytes
else
        defs    512-$
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 NextZXOS 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 please consult the list of existing
; driver ids (c:/docs/nextzxos/DriverIDs.txt) for information on obtaining
; a suitable id for your driver.


        org     $0000

        defm    "NDRV"          ; .DRV file signature

        defb    $01+$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    $80+$01         ; number of additional 8K DivMMC RAM banks
                                ; required (0..8); call init/shutdown
        ; NOTE: If bit 7 of the "mmcbanks" value above is set, .INSTALL and
        ;       .UNINSTALL will call your driver's $80 and $81 functions
        ;       to allow you to perform initialisation/shutdown tasks
        ;       (see border.asm for more details)

        defb    3               ; 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..8192)
;                                 ; (remaining space will be erased to zeroes)
;        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 by your
;              driver code, 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..8192)
;                                 ; (remaining space will be erased to zeroes)
;        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 by your
;              driver code, as .uninstall will use the value for deallocating.


; Although our simple driver doesn't actually need any additional memory banks,
; we have requested 1 DivMMC bank and 3 Spectrum RAM banks as an example.

; First, the 1 DivMMC bank that was requested:

        defb    1                        ; 1 patch
        defw    0                        ; no data to be preloaded into this bank
                                         ; (it will be erased to zeroes)
        ; List of patches to be replaced with this bank's id
        defw    bankid_mmc0              ; offset in driver to patch the bank id

; Then the 3 Spectrum RAM banks that were requested:

; First bank:
        defb    1                        ; 1 patch
        defw    b0data_end-b0data        ; size of preload data

        ; The actual preloaded data follows (the remainder of the 8K bank will
        ; be erased to zeroes)
b0data:
        defs    800,$aa                  ; 800 bytes filled with $AA
        defm    "This is the first allocated ZX bank"
        defs    20,$55                   ; 20 bytes filled with $55
b0data_end:
        ; List of patches to be replaced with this bank's id
        defw    bankid_zx0              ; offset in driver to patch the bank id


; Second bank:
        defb    1                        ; 1 patch
        defw    0                        ; no data to be preloaded into this bank
                                         ; (it will be erased to zeroes)
        ; List of patches to be replaced with this bank's id
        defw    bankid_zx1              ; offset in driver to patch the bank id


; Third bank:
```

```
        defb    1                       ; 1 patch
        defw    b2data_end-b2data       ; size of preload data

        ; The actual preloaded data follows (the remainder of the 8K bank will
        ; be erased to zeroes)
b2data:
        defm    "This is the third allocated ZX bank"
b2data_end:
        ; List of patches to be replaced with this bank's id
        defw    bankid_zx2              ; offset in driver to patch the bank id
```

Printer driver example (file 1 of 2) - sample_prt.asm

```
; ****************************************************************************
; * Example NextZXOS printer driver                                          *
; ****************************************************************************
;
;
; This file is the 512-byte NextZXOS driver itself, plus relocation table.
;
;
; Assemble with: pasmo sample_prt.asm sample_prt.bin sample_prt.sym
;
;
; After this, sample_prt_drv.asm needs to be built to generate the actual
; driver file.
;
;
; GENERAL NOTES ON PRINTER/AUX DRIVERS:
;
;
; A printer driver should use "P" as its driver id. This allows the user
; to install whatever printer driver is appropriate for them, and for
; software to use it in a standardised way.
;
;
; NextZXOS contains a built-in "P" driver suitable for the ZX Printer,
; Alphacom 32 and Timex 2040. This will be overridden by any user-installed
; "P" driver for alternative printers.
;
;
; In particular, NextBASIC will automatically send data LPRINT/LLISTed
; (or PRINTed to #3, or any other stream that has been opened to
; BASIC channel "P") to any installed driver with id "P".
;
;
; Similarly, if a "P" driver has been installed, CP/M will use this for
; output to its logical LST: device (also referred to as the physical
; LPT device).
;
;
; In order to support NextBASIC and CP/M, a printer driver only needs to
; support the standard calls $f7 (return output status) and $fb (output
; character).
;
;
; Optionally, you can support the COPY command which uses call $f6. The
; built-in ZX printer driver always copies the standard Spectrum screen
; in this case, even if other modes such as layer 2 are active. Drivers for
; more capable printers may want to support colour copies, layer2, hi-res,
; lo-res, tilemaps etc. Note that if you are doing this, you should use the
; state of the appropriate Next ports and registers to determine which screen
; is currently active (eg the Layer2 and Timex ports, and the Sprites/Layers
; and Tilemap next registers). Don't rely on system variable information,
; since this call may be made when system variables are not valid (eg from
; the Multiface).
;
;
; You may of course support any other standard calls that;
; CP/M also supports an AUX physical device (with default input/output
; through the logical AUXIN: and AUXOUT: devices). This will
; automatically be routed to any installed driver with id "X".
; An AUX driver can be written in the same way as a printer driver, but
; should additionally support standard calls $f8 (return input status)
; and $fc (input character).
;
;
; See the example border.asm/border_drv.asm driver if your driver needs to
; be run on the IM1 interrupt, or if it needs additional 8K DivMMC/ZX RAM
; banks. This sample printer driver (and probably most printer drivers) do not
; require these, so discussion of them is not present in the example
; printer driver.


; ****************************************************************************
;
```

```
; * Definitions                                                           *
; ************************************************************************
;
; The port used by our hypothetical printer. Don't try and use this driver
; as it won't do anything!

printer_port    equ     $ff


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

        org     $0000

; At $0000 is the entry point for API calls directed to the printer
; driver.
; NOTE: If your printer driver needs to be called on the IM1 interrupt
;       you will need to provide an entry point at $0003 for this (see
;       border.asm example driver for full details).
;       This simple printer driver doesn't need interrupts so there is
;       no need to provide the $0003 entry point.

api_entry:

; On entry, B=call id with HL,DE other parameters.
; You may provide any standard or driver-specific calls that you wish.
; See the example border.asm driver for a description of the standard calls.
; However, a standard printer driver that supports NextBASIC and CP/M only
; needs to provide 2 standard calls:
;   B=$f7: return output status
;   B=$fb: output character

        ld      a,b
        cp      $fb                         ; "output character" call?
        jr      z,output_char               ; on if so
        cp      $f7                         ; "return output status" call?
        jr      z,return_status             ; on if so

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


; ************************************************************************
;
; * Return output status ($f7)                                           *
; ************************************************************************
;
; This call is entered with D=handle.
; CP/M always calls with D=1 (system handle) and a printer
; driver can generally ignore the handle id unless you support standard
; calls for opening/closing multiple different streams and wish them all
; to be handled independently.
; This call should return with carry clear to indicate success and
; BC=$ffff if the printer is ready to accept a character for output, or
; BC=$0000 if the printer is not ready.

; Our hypothetical printer interface has a BUSY signal connected to bit
; 0 of the input data on the printer port, so we will check this and
; return the status accordingly.

return_status:
        ld      bc,$ffff
```

```
        and     a                       ; clear carry to indicate success
        in      a,(printer_port)        ; get signals from printer
        bit     0,a                     ; check BUSY signal
        ret     z                       ; exit with BC=$ffff if not busy
        inc     bc
        ret                             ; exit with BC=$0000 if bust


; **************************************************************************
; * Output character ($fb)                                                 *
; **************************************************************************
;
; This call is entered with D=handle and E=character.
; NextBASIC and CP/M always call with D=1 (system handle) and a printer
; driver can generally ignore the handle id unless you support standard
; calls for opening/closing multiple different streams and wish them all
; to be handled independently.
; This call should return with carry clear to indicate success.
; If you return with carry set and A=$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.

output_char:
        ; It's good practice to allow the user to abort with BREAK if
        ; the printer is stuck in a busy loop.
        ld      a,$7f
        in      a,($fe)
        rra
        jr      c,check_printer         ; on if SPACE not pressed
        ld      a,$fe
        in      a,($fe)
        rra
        jr      c,check_printer         ; on if CAPS SHIFT not pressed
        ld      a,$fe                   ; exit with A=$fe and carry set
        scf                             ; so "End of file" reported
        ret
check_printer:
        ; Wait for the printer to become ready.
        in      a,(printer_port)        ; get signals from printer
        bit     0,a                     ; check BUSY signal
        jr      nz,output_char          ; loop back if printer is busy
        ld      a,e                     ; A=character to output
        out     (printer_port),a        ; send to the printer
        and     a                       ; clear carry to indicate success
        ret


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

if ($ > 512)
.ERROR Driver code exceeds 512 bytes
else
        defs    512-$
endif

; Each relocation is the offset of the high byte of an address to be relocated.
; This particular driver is so simple it doesn't contain any absolute addresses
; needing to be relocated. (border.asm is a slightly more complex driver that
```

```
; does have a relocation table).

reloc_start:
reloc_end:
```

Printer driver example (file 2 of 2) - sample_prt_drv.asm

```
; ****************************************************************************
; * Example NextZXOS printer driver file                                    *
; ****************************************************************************
;
;
; This file generates the actual sample_prt.drv file which can be installed or
; uninstalled using the .install/.uninstall commands.
;
; The driver itself (sample_prt.asm) must first be built.
;
; Assemble this file with: pasmo sample_prt_drv.asm sample_prt.drv
;
; GENERAL NOTES ON PRINTER/AUX DRIVERS:
;
; A printer driver should use "P" as its driver id. This allows the user
; to install whatever printer driver is appropriate for them, and for
; software to use it in a standardised way.
;
; In particular, NextBASIC will automatically send data LPRINT/LLISTed
; (or PRINTed to #3, or any other stream that has been opened to
; BASIC channel "P") to any installed driver with id "P".
;
; Similarly, if a "P" driver has been installed, CP/M will use this for
; output to its logical LST: device (also referred to as the physical
; LPT device).
;
; In order to support NextBASIC and CP/M, a printer driver only needs to
; support the standard calls $f7 (return output status) and $fb (output
; character). You may of course support any other standard calls that
; you like (or additional driver-specific calls, for example to set the
; communications parameters for a serial printer).
;
; CP/M also supports an AUX physical device (with default input/output
; through the logical AUXIN: and AUXOUT: devices). This will
; automatically be routed to any installed driver with id "X".
; An AUX driver can be written in the same way as a printer driver, but
; should additionally support standard calls $f8 (return input status)
; and $fc (input character).
;
; See the example border.asm/border_drv.asm driver if your driver needs to
; be run on the IM1 interrupt, or if it needs additional 8K DivMMC/ZX RAM
; banks. This sample printer driver (and probably most printer drivers) do not
; require these, so discussion of them is not present in the example
; printer driver.


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

        include "sample_prt.sym"

relocs  equ     (reloc_end-reloc_start)/2


; ****************************************************************************
; * .DRV file header                                                        *
; ****************************************************************************
;
```

```
        org     $0000

        defm    "NDRV"          ; .DRV file signature

        defb    "P"             ; standard driver id for printer device.

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

        defb    0               ; number of 8K DivMMC RAM banks needed
        defb    0               ; number of 8K Spectrum RAM banks needed


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

        incbin  "sample_prt.bin"
```

```
; **************************************************************************
; * Example NextZXOS keyboard driver                                       *
; **************************************************************************
;
; The keyboard driver used by NextZXOS may be replaced by installing a
; special driver with id 0.

; This file is the 512-byte NextZXOS driver itself, plus relocation table.
;
; Assemble with: pasmo keyboard.asm keyboard.bin keyboard.sym
;
; After this, keyboard_drv.asm needs to be built to generate the actual
; driver file.

; Keyboard drivers are installed using the same .install dot command
; as standard drivers, and immediately replace the existing keyboard
; driver (the keyboard driver does not count towards the total number
; of standard installable NextZXOS drivers).
;
; The main differences between the keyboard driver and standard drivers
; are as follows:
;       1. The keyboard driver always has driver id 0.
;       2. The keyboard driver cannot provide an API.
;       3. The keyboard driver is always called at every IM1 interrupt.
;       4. The keyboard driver has just a single entry point, at $0000,
;          which is called during IM1 interrupts.
;
; Replacement keyboard drivers should perform the same effective
; functionality as the standard KEYBOARD routine at $02bf in the ROM of
; the original 48K Spectrum.
;
; The following driver replicates the code from the original
; ROM (although slightly re-ordered). It additionally reads the Kempston
; joystick port so a joystick may be used for navigation purposes within
; NextZXOS. It may be used as a base for a replacement driver.
;
; Possible uses for replacement keyboard drivers might be:
;       * For use with alternative international keyboard layouts
;       * Adding a multi-byte buffer to allow faster typing
;
; Be aware that the driver is called by all ROMs, so should support
; keyword tokens (unless you don't intend to use 48K BASIC mode, or only
; intend to use 48K BASIC mode using the Gosh Wonderful ROM in standard
; single-letter entry).


; **************************************************************************
; * System variable definitions                                           *
; **************************************************************************
;

KSTATE  equ     $5c00
LAST_K  equ     $5c08
REPDEL  equ     $5c09
REPPER  equ     $5c0a
ERR_NR  equ     $5c3a
FLAGS   equ     $5c3b
MODE    equ     $5c41
FLAGS2  equ     $5c6a


iy_FLAGS        equ     FLAGS-ERR_NR
```

```
iy_MODE           equ     MODE-ERR_NR
iy_FLAGS2         equ     FLAGS2-ERR_NR


; ***************************************************************************
;
; * Next register definitions                                               *
; ***************************************************************************
;

next_reg_select                           equ     $243b
nxr_core_boot                             equ     $10


; ***************************************************************************
;
; * Token definitions                                                       *
; ***************************************************************************
;

udgA                      equ     144
token_sin                 equ     $b2
token_to                  equ     $cc
token_new                 equ     $e6


; ***************************************************************************
;
; * KEYBOARD routine (at $02bf in original 48K ROM)                         *
; ***************************************************************************
;

        org     $0000           ; this is the entry point for the driver

keyboard:
reloc_1:
        call    key_scan                ; scan the matrix
        ret     nz                      ; ignore invalid key combinations
        ld      hl,KSTATE               ; start with KSTATE0..3
keyboard_2:
        bit     7,(hl)
        jr      nz,keyboard_3           ; on if the set is free
        inc     hl
        dec     (hl)                    ; otherwise decrease its counter
        dec     hl
        jr      nz,keyboard_3
        ld      (hl),$ff                ; signal set is free after 5 calls
keyboard_3:
        ld      a,l
        ld      hl,KSTATE+$04
        cp      l
        jr      nz,keyboard_2           ; back with 2nd set (KSTATE4..7)
reloc_2:
        call    k_test                  ; check for a valid key

; NOTE: At this point, the driver in the original ZX ROM simply returned
;       if no key is pressed (carry clear). In the NextZXOS driver, we
;       additionally check for the Kempston joystick.

        jr      c,lk_gotkey             ; on if valid key
        in      a,($1f)                 ; else read kempston port
        cp      $ff
        ret     z                       ; exit if $ff (no Kempston port)
        and     a
        ret     z                       ; exit if $00
reloc_13:
        ld      hl,kempston_keys-1
kemp_decode_loop:
```

```
        inc     hl                      ; next table address
        srl     a                       ; next port bit to carry
        jr      nc,kemp_decode_loop     ; until found a set bit
        ret     nz                      ; exit if more than one set bit
        ld      a,(hl)                  ; A=code

; The standard ZX ROM keyboard routines now continue.

lk_gotkey:
        ld      hl,KSTATE               ; start with KSTATE0..3
        cp      (hl)
        jr      z,k_repeat              ; on for repeat if matching key
        ex      de,hl
        ld      hl,KSTATE+$04           ; again with KSTATE4..7
        cp      (hl)
        jr      z,k_repeat              ; on for repeat of 2nd set's key
        bit     7,(hl)
        jr      nz,keyboard_4           ; on with new key if 2nd set is free
        ex      de,hl
        bit     7,(hl)
        ret     z                       ; exit if neither set is free
keyboard_4:
        ld      e,a
        ld      (hl),a                  ; store "main" key in KSTATE0/4
        inc     hl
        ld      (hl),$05                ; initialise counter in KSTATE1/5
        inc     hl
        ld      a,(REPDEL)
        ld      (hl),a                  ; initialise repeat delay in KSTATE2/6
        inc     hl
        ld      c,(iy+iy_MODE)          ; C=(MODE) system variable
        ld      d,(iy+iy_FLAGS)         ; D=(FLAGS) system variable
        push    hl
reloc_3:
        call    k_decode                ; fully decode key with current shifts
        pop     hl
        ld      (hl),a                  ; store decoded key in KSTATE 3/7
keyboard_5:
        ld      (LAST_K),a              ; update last key
        set     5,(iy+iy_FLAGS)         ; and signal a new key is available
        ret


; ****************************************************************************
;
; * Kempston key translation table                                          *
; ****************************************************************************
;

kempston_keys:
        defb    9                       ; cursor right
        defb    8                       ; cursor left
        defb    10                      ; cursor down
        defb    11                      ; cursor up
        defb    13                      ; fire (ENTER)
        defb    32                      ; button 2 (SPACE)
        defb    7                       ; button 3 (EDIT)
        defb    7                       ; button 4 (EDIT)


; ****************************************************************************
;
; * K-REPEAT routine (at $0310 in original 48K ROM)                         *
; ****************************************************************************
;
; Enters with HL pointing to the repeating set (KSTATE0 or KSTATE4)
```

```
k_repeat:
        inc     hl
        ld      (hl),$05                ; reset the set's 5-call counter
        inc     hl
        dec     (hl)                    ; decrement the repeat delay
        ret     nz                      ; exit if too early to repeat
        ld      a,(REPPER)
        ld      (hl),a                  ; set delay for further repeats
        inc     hl
        ld      a,(hl)                  ; fetch the fully-decoded key
        jr      keyboard_5              ; and register it as a new press


; ***************************************************************************
; * Keytables                                                               *
; ***************************************************************************
;
; These are copies of the key tables from original 48K ROM

; The L-mode keytable with CAPS-SHIFT

keytable_l:
        defm    "BHY65TGV"
        defm    "NJU74RFC"
        defm    "MKI83EDX"
        defm    $0e,"LO92WSZ"
        defm    " ",$0d,"P01QA"

; The extended-mode keytable (unshifted letters)

keytable_e:
        defb    $e3,$c4,$e0,$e4
        defb    $b4,$bc,$bd,$bb
        defb    $af,$b0,$b1,$c0
        defb    $a7,$a6,$be,$ad
        defb    $b2,$ba,$e5,$a5
        defb    $c2,$e1,$b3,$b9
        defb    $c1,$b8

; The extended mode keytable (shifted letters)

keytable_e_s:
        defb    $7e,$dc,$da,$5c
        defb    $b7,$7b,$7d,$d8
        defb    $bf,$ae,$aa,$ab
        defb    $dd,$de,$df,$7f
        defb    $b5,$d6,$7c,$d5
        defb    $5d,$db,$b6,$d9
        defb    $5b,$d7

; The control code keytable (CAPS-SHIFTed digits)

keytable_cc:
        defb    $0c,$07,$06,$04
        defb    $05,$08,$0a,$0b
        defb    $09,$0f

; The symbol code keytable (letters with symbol shift)

keytable_sym:
        defb    $e2,$2a,$3f,$cd
        defb    $c8,$cc,$cb,$5e
```

```
        defb    $ac,$2d,$2b,$3d
        defb    $2e,$2c,$3b,$22
        defb    $c7,$3c,$c3,$3e
        defb    $c5,$2f,$c9,$60
        defb    $c6,$3a


; The extended mode keytable (SYM-SHIFTed digits)


keytable_e_d:
        defb    $d0,$ce,$a8,$ca
        defb    $d3,$d4,$d1,$d2
        defb    $a9,$cf



; ***************************************************************************
; * KEY-SCAN routine (at $028e in original 48K ROM)                         *
; ***************************************************************************
;
; Scans the keyboard matrix and returns with:
;       E=1st keycode ($ff=none)
;       D=2nd (shift) keycode ($ff=none)
;       zero flag set if invalid (3+ keys pressed, or 2 non-shifts pressed)
; If both shift keys are pressed, D contains CAPS ($27) and E contains SYM ($18)


key_scan:
        ld      l,$2f                   ; initial value for each row: $2f..$28
        ld      de,$ffff                ; D=E=no key
        ld      bc,$fefe                ; C=port, B=row
key_scan_2:
        in      a,(c)                   ; read row
        cpl
        and     $1f
        jr      z,key_scan_5            ; on if no keys pressed in row
        ld      h,a                     ; H=row bits
        ld      a,l                     ; A=initial key value
key_scan_3:
        inc     d                       ; exit with zero reset if this is the
        ret     nz                      ; third key to be detected
key_scan_4:
        sub     $08                     ; find keycode value for each bit
        srl     h                       ; in turn
        jr      nc,key_scan_4
        ld      d,e                     ; copy any previous keycode to D
        ld      e,a                     ; E=keycode
        jr      nz,key_scan_3           ; back if further keys in this row
key_scan_5:
        dec     l                       ; next row's initial keycode
        rlc     b                       ; next row port value
        jr      c,key_scan_2            ; back until all rows scanned
        ld      a,d
        inc     a
        ret     z                       ; exit with zero set if 0 or 1 keys
        cp      $28
        ret     z                       ; or if 2nd key of pair is CAPS SHIFT
        cp      $19
        ret     z                       ; or if 2nd key of pair is SYM SHIFT
        ld      a,e                     ; SYM SHIFT could be 1st key of a pair
        ld      e,d                     ; so swap them
        ld      d,a
        cp      $18                     ; and exit with zero set if so
        ret                             ; or zero reset for an invalid pair
```

```
; ******************************************************************************
; * K-TEST routine (at $031e in original 48K ROM)                              *
; ******************************************************************************
;
; Given a pair of keycodes in D and E, returns with carry set and
; A containing the decoded L-mode key and B containing the shift keycode.
; Returns with carry reset if no key or only a single shift is pressed
; (if both shifts are pressed, the decoded value in A is $0e, EXTEND).

        k_test:
                ld      b,d                     ; B=shift keycode ($ff/$27/$18)
                ld      d,$00
                ld      a,e
                cp      $27                     ; return with carry reset if the
                ret     nc                      ; 1st key was CAPS or "no key"
                cp      $18
                jr      nz,k_test_2
                bit     7,b                     ; return with carry reset if
                ret     nz                      ; only SYM shift pressed
        k_test_2:
        reloc_4:
                ld      hl,keytable_l   ; the main keytable
                add     hl,de
                ld      a,(hl)                  ; A=decoded L-mode key
                scf                             ; carry set, valid key
                ret


; ******************************************************************************
; * K-DECODE routine (at $0333 in original 48K ROM)                            *
; ******************************************************************************
;
; Enters with:
;       C=contents of MODE system variable
;       D=contents of FLAGS system variable
;       E="main" key code (assuming L mode and no shifts)
;       B=shift key code ($ff/$27/$18)

        k_decode:
                ld      a,e
                cp      '9'+1
                jr      c,k_decode_6            ; on for digits, SPACE, ENTER, EXTEND
                dec     c
        reloc_5:
                jp      m,k_decode_4            ; on for letters in K/L/C modes
                jr      z,k_decode_2            ; on for letters in E mode

                ; Letters in G mode
                ld      d,b                     ; D=shift key
                ld      bc,next_reg_select
                ld      e,nxr_core_boot
                out     (c),e
                inc     b
                in      e,(c)
                add     a,token_to-'A'          ; DRIVE+letters to TO..RESTORE
                bit     1,e
                ret     nz
                add     a,udgA-token_to         ; unshifted letters to UDGs
                inc     d
                ret     z
                add     a,token_new-udgA        ; CAPS-shifted letters to NEW..COPY
                bit     0,d
                ret     z
                add     a,token_sin-token_new   ; SYM-shifted letters to SIN..THEN
```

```
        ret

        ; Letters in E mode
k_decode_2:
reloc_6:
        ld      hl,keytable_e-'A'       ; unshifted E-mode table
        inc     b
        jr      z,k_decode_3            ; on if neither shift pressed
reloc_7:
        ld      hl,keytable_e_s-'A'     ; shifted E-mode table

        ; Enter here with:
        ;       E="main" key
        ;       HL=keytable offset by minimum value of E
k_decode_3:
        ld      d,$00
        add     hl,de                   ; index into table
        ld      a,(hl)                  ; A=fully-decoded key
        ret

        ; Letters in K/L/C modes
k_decode_4:
reloc_8:
        ld      hl,keytable_sym-'A'     ; symbol code table
        bit     0,b
        jr      z,k_decode_3            ; back for SYM-SHIFTed letters
        bit     3,d
        jr      z,k_decode_5            ; on for K mode
        bit     3,(iy+iy_FLAGS2)
        ret     nz                      ; exit with upper-case letter in C mode
        inc     b
        ret     nz                      ; or if CAPS SHIFT held
        add     a,'a'-'A'               ; otherwise convert to lower-case
        ret

        ; Letters in K mode
k_decode_5:
        add     a,token_new-'A'         ; convert letter to keyword tokens
        ret

        ; Digits, SPACE, ENTER, EXTEND
k_decode_6:
        cp      '0'
        ret     c                       ; no conversion for SPACE/ENTER/EXTEND
        dec     c
reloc_9:
        jp      m,k_decode_9            ; on for K/L/C modes
        jr      nz,k_decode_8           ; on for G mode
reloc_10:
        ld      hl,keytable_e_d-'0'     ; extended mode table for digits
        bit     5,b
        jr      z,k_decode_3            ; use table if SYM SHIFT held
        cp      '8'
        jr      nc,k_decode_7           ; on for '8' or '9'
        sub     $20                     ; convert to "paper colour" $10..$17
        inc     b                       ; if no shift held
        ret     z
        add     a,$08                   ; convert to "ink colour" $18..$1f
        ret                             ; if CAPS SHIFT held

        ; '8' or '9' in E mode with CAPS SHIFT or no shift
k_decode_7:
```

```
        sub     '6'                     ; convert to "bright code" $02..$03
        inc     b                       ; if no shift held
        ret     z
        add     a,$fe                   ; convert to "flash code" $00..$01
        ret                             ; if CAPS SHIFT held

        ; Digits in G mode
k_decode_8:
reloc_11:
        ld      hl,keytable_cc-'0'      ; control code keytable
        cp      $39
        jr      z,k_decode_3            ; special case 9 (GRAPHICS mode)
        cp      $30
        jr      z,k_decode_3            ; special case 0 (DELETE)
        add     a,170-'1'               ; SYM-shifted digits to 170..177
        bit     0,b
        ret     z
        dec     a                       ; unshifted digits to 128..135
        and     $07
        or      $80
        inc     b
        ret     z
        xor     $0f                     ; CAPS-shifted digits to 136..143
        ret

        ; Digits in K/L/C modes
k_decode_9:
        inc     b                       ; use digits if unshifted
        ret     z
        bit     5,b
reloc_12:
        ld      hl,keytable_cc-'0'      ; use control code table if CAPS SHIFTed
        jr      nz,k_decode_3
        ; Symbol-shifted digits
        sub     $10                     ; convert to: SPACE,!,",#,$,%,&,',(,)
        cp      '"'                     ; " needs to be replaced with @
        jr      z,k_decode_10
        cp      ' '                     ; SPACE needs to be replaced with _
        ret     nz
        ld      a,'_'                   ; SYM SHIFT+0 is _
        ret
k_decode_10:
        ld      a,'@'                   ; SYM SHIFT+2 is @
        ret


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


if ($ > 512)
.ERROR Driver code exceeds 512 bytes
else
        defs    512-$
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+2
        defw    reloc_4+2
        defw    reloc_5+2
        defw    reloc_6+2
        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
reloc_end:
```

Keyboard driver example (file 2 of 2) - keyboard_drv.asm

```
; *************************************************************************
; * Example NextZXOS keyboard driver file                                *
; *************************************************************************
;
;
; This file generates the actual keyboard.drv file which can be installed
; using the .install command, to replace the built-in keyboard driver.
;
; The driver itself (keyboard.asm) must first be built.
;
; Assemble this file with: pasmo keyboard_drv.asm keyboard.drv


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

        include "keyboard.sym"

relocs  equ     (reloc_end-reloc_start)/2


; *************************************************************************
; * .DRV file header                                                     *
; *************************************************************************
;
; The keyboard driver id is always zero (bit 7 may be set but will always be
; treated as if it is set, since the keyboard driver is always called on
; interrupts).

        org     $0000

        defm    "NDRV"          ; .DRV file signature

        defb    $00             ; keyboard driver id

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

        defb    0               ; number of additional 8K DivMMC RAM banks

        defb    0               ; number of additional 8K Spectrum RAM banks


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

        incbin  "keyboard.bin"
```

List of updates

Updates: 24 May 2023

Updated IDE_WINDOW_LINEIN call with new facility for auto-rewind.

Updates: 14 Oct 2022

Removed obsolete caveats about esxDOS being unable to access +3DOS-formatted filesystems, since it can now do this (from v2.07).

Updates: 26 Sep 2022

Clarifications in the streaming API notes.

Updated example keyboard driver.

Updated IDE_WINDOW_LINEIN call with new flag for allowing token/UDG entry.

Updates: 18 Jun 2022

Updated DOS_OPEN, F_UNLINK and F_RENAME with metadata information.

Updates: 26 May 2022

Clarified drive specifier usage in each relevant call.

Updates: 31 March 2022

Removed obsolete note about layer2.

Updates: 26 February 2022

Added documentation for newly-implemented M_AUTOLOAD esxDOS hook.

Removed comment about "usual" drivers DivMMC RAM paging value as this is no longer accurate.

Updates: 17 February 2022

Noted the M_TAPEIN/M_TAPEOUT now always return '*' as the esxDOS drive specifier.

Clarified data returned by F_STAT/F_FSTAT.

Clarified default sorting in DOS_CATALOG.

Updated IDE_BROWSER with additional colour scheme features.

Updated DOS_OPEN with new shared-write and shared-read-write access modes.

Updated example driver sources to match standalone source files (in src/asm/).

Fixed typos.

Updates: 19 September 2021

Updated IDE_SNAPLOAD notes.

Updated IDE_BROWSER, DOS_CATALOG, F_OPENDIR and F_READDIR descriptions to include the new sorting, filtering and preferences features added in NextZXOS v2.07.

Updated IDE_WINDOW_LINEIN with the new functionality provided in NextZXOS v2.07.

Updated IDE_SWAP_OPEN to note that any unfragmented file can now be specified as a swap file.

Updated IDE_DOS_MAPPING with new alternative usage.

Updates: 26 October 2020

Clarifications to IDE_BANK, IDE_WINDOW_LINEIN and IDE_MODE notes.

Clarifications to stack requirements when using M_P3DOS.

Updates: 28 July 2020

Clarifications about driver IDs.

Clarified use of DISK_FILEMAP in stream.asm example.

Removed duplicated entry for IDE_MOUNT.

Updates: 25 March 2020

Updated lists of supported +3DOS calls, as v2.06+ now support various low-level calls for use with legacy software running from .DSK images.

Updates: 30 January 2020

Updated NextZXOS version number referred to, for day zero release (v2.04).

Updates: 1 Dec 2019
Corrected outputs of IDE_RTC and M_GETDATE to include the values returned in HL.

Updates: 4 Nov 2019
Fixed incorrect mentions of "extensions" in the text; these are properly referred to as "types".

Updates: 4 Aug 2019
Clarified that F_CHDIR does not change the current drive.

Updates: 17 July 2019
Added reminder in each individual esxDOS hook that IX entry parameters need to instead be passed in HL from a dot command.

Updates: 14 July 2019
Clarified notes on stack usage in M_P3DOS.

Added clarifying notes for IDE_BASIC call.

Added new IDE_TOKENISER call.

Updates: 18 May 2019
Added clarifying notes to DISK_FILEMAP and M_DRVAPI hooks.

Updates: 11 Jan 2019
Added Kempston joystick support to sample keyboard driver.

Clarified that some calls in the +3DOS API require IY to be unchanged from the standard value set by BASIC (ERR_NR, $5c3a).

Updates: 22 Sep 2018
Added more notes on usage of IDE_BROWSER.

Added option to DISK_STRMSTART to avoid the initial wait for the data token, to allow the user to cover the latency (from v2.01).

Updates: 8 Sep 2018
Added option for F_OPENDIR/F_READDIR to use wildcards.

Updates: 2 Sep 2018
Noted that the IDE_MODE call also returns the mode window handle in IX.

Updates: 30 Aug 2018
Enhanced F_GETCWD call to additionally allow a consistently named path to be returned for any supplied filespec.

Noted that M_GETSETDRV now ignores the lower 3 bits, so these can be used to ensure A<>0 if needing to set the current drive.

Clarified entry parameters for M_P3DOS.

Updates: 25 Aug 2018
The M_TAPEIN hook with B=in_flags now also returns the original setting of the flags in A.

Clarified that command-lines for dot commands may include ':' as part of the line if enclosed within double-quotes.

Updates: 24 Aug 2018
Added new IDE_MODE call in the +3DOS API for querying the current NextBASIC display mode setting, or changing mode.

Updates: 19 Aug 2018
Noted that dot commands are now loaded from C:/DOT rather than C:/BIN. This makes it easier for esxDOS and NextZXOS to co-exist.

Updates: 17 Aug 2018
Added new IDE_MOUNT call in the +3DOS API for unmounting/remounting SD cards.

Updates: 15 Aug 2018
Added new reason to the system driver API used by .UNINSTALL.
Added more information to the driver section.
Updated example driver (border.asm and border_drv.asm) with sample code for bank allocation and usage.
Added sample printer driver.

Updates: 6 Aug 2018
Updated **M_DOSVERSION** which now additionally provides ROM language information.
Updated lists of deprecated +3DOS/IDEDOS calls.

Updates: 31 Jul 2018
Added further notes on **IDE_SNAPLOAD**.

Updates: 15 Jul 2018
Added BROWSERCAPS_UNMOUNT capability to **IDE_BROWSER**.

Updates: 13 Jul 2018
Added new rc_fragmented error code.
Added new options to **IDE_SWAP_OPEN** to allow any named (unfragmented) file to be opened and used as a swap file.
Deprecated **IDE_SWAP_EX**.
Noted that **DOS_READ/DOS_WRITE/IDE_SWAP_IN/IDE_SWAP_OUT** can take any valid 16K RAM page number 0-111, not just 0-7 as on the +3/+3e.

Updates: 8 Jul 2018
Updated IDE_DOS_MAP and IDE_DOS_MAPPING calls with special device $ff indicating a mounted filesystem image (.P3D or .DSK file).

Updates: 28 Jun 2018
Added new rc_bank_available reason to the IDE_BANK call.
Added note about the order of bank allocation.

Updates: 26 Jun 2018
Added M_SETCAPS hook to modify behaviour of other calls. Currently allows F_FTRUNCATE/F_TRUNCATE to be sped up by omitting to zeroise any new file sections.

Updates: 8 Jun 2018
The IDE_BROWSER call now returns the address of an LFN (in DE), as well as the short name (in HL).

Updates: 12 Mar 2018
Updated driver API to allow an additional driver for the keyboard to be installed, replacing the standard keyboard driver.

Added example keyboard driver (keyboard.asm & keyboard_drv.asm).

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.