NextOS API (Updated 12 Dec 2017)

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

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.

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

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

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

DOS_VERSION ($0103)	Get +3DOS issue and version numbers
*DOS_OPEN ($0106)	Create and/or open a file
DOS_CLOSE ($0109)	Close a file
DOS_ABANDON ($010C)	Abandon a file
DOS_REF_HEAD ($010F)	Point at the header data for this file
DOS_READ ($0112)	Read bytes into memory
DOS_WRITE ($0115)	Write bytes from memory
DOS_BYTE_READ ($0118)	Read a byte
DOS_BYTE_WRITE ($011B)	Write a byte

```
*DOS_CATALOG ($011E)          Catalog disk directory
*DOS_FREE_SPACE ($0121)       Free space on disk
DOS_DELETE ($0124)            Delete a file
DOS_RENAME ($0127)            Rename a file
DOS_BOOT ($012A)              Boot an operating system or other program
DOS_SET_DRIVE ($012D)         Set/get default drive
DOS_SET_USER ($0130)          Set/get default user number
*DOS_GET_POSITION ($0133)     Get file pointer for random access
DOS_SET_POSITION ($0136)      Set file pointer for random access
*DOS_GET_EOF ($0139)          Get end of file position for random access
DOS_GET_1346 ($013C)          Get memory usage in pages 1, 3, 4, 6
DOS_SET_1346 ($013F)          Re-allocate memory usage in pages 1, 3, 4, 6
DOS_FLUSH ($0142)             Bring disk up to date
DOS_SET_ACCESS ($0145)        Change open file's access mode
DOS_SET_ATTRIBUTES ($0148)    Change a file's attributes
DOS_SET_MESSAGE ($014E)       Enable/disable error messages

IDE_VERSION ($00A0)           Get IDEDOS version number
IDE_SWAP_OPEN ($00D9)         Open a swap partition
IDE_SWAP_CLOSE ($00DC)        Close a swap partition
IDE_SWAP_OUT ($00DF)          Write block to swap partition
IDE_SWAP_IN ($00E2)           Read block from swap partition
IDE_SWAP_EX ($00E5)           Exchange block with swap partition
IDE_SWAP_POS ($00E8)          Get current block number in swap partition
IDE_SWAP_MOVE ($00EB)         Set current block number in swap partition
IDE_SWAP_RESIZE ($00EE)       Change block size of swap partition
IDE_PARTITION_FIND ($00B5)    Find named partition
*IDE_DOS_MAP ($00F1)          Map drive to partition
*IDE_DOS_UNMAP ($00F4)        Unmap drive
IDE_DOS_MAPPING ($00F7)       Get drive mapping
*IDE_SNAPLOAD ($00FD)         Load a snapshot
*IDE_PATH ($01b1)             Create, delete, change or get directory
%IDE_CAPACITY ($01b4)         Get card capacity
%IDE_GET_LFN ($01b7)          Get long filename
%IDE_BROWSER ($01ba)          File browser
```

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

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

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

```
DOS_REF_XDPB ($0151)          Point at XDPB for low level disk access
DOS_MAP_B ($0154)             Map B: onto unit 0 or 1
DD_INTERFACE ($0157)          Is the floppy disk driver interface present?
DD_INIT ($015A)               Initialise disk driver
DD_SETUP ($015D)              Specify drive parameters
DD_SET_RETRY ($0160)          Set try/retry count
DD_READ_SECTOR ($0163)        Read a sector
```

DD_WRITE_SECTOR ($0166)	Write a sector
DD_CHECK_SECTOR ($0169)	Check a sector
DD_FORMAT ($016C)	Format a track
DD_READ_ID ($016F)	Read a sector identifier
DD_TEST_UNSUITABLE ($0172)	Test media suitability
DD_LOGIN ($0175)	Log in disk, initialise XDPB
DD_SEL_FORMAT ($0178)	Pre-initialise XDPB for DD FORMAT
DD_ASK_1 ($017B)	Is unit 1 (external drive) present?
DD_DRIVE_STATUS ($017E)	Fetch drive status
DD_EQUIPMENT ($0181)	What type of drive?
DD_ENCODE ($0184)	Set intercept routine for copy protection
DD_L_XDPB ($0187)	Initialise an XDPB from a disk specification
DD_L_DPB ($018A)	Initialise a DPB from a disk specification
DD_L_SEEK ($018D)	uPD765A seek driver
DD_L_READ ($0190)	uPD765A read driver
DD_L_WRITE ($0193)	uPD765A write driver
DD_L_ON_MOTOR ($0196)	Motor on, wait for motor-on time
DD_L_T_OFF_MOTOR ($0199)	Start the motor-off ticker
DD_L_OFF_MOTOR ($019C)	Turn the motor off

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

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

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

DOS_OPEN_DRIVE ($014B)	Open a drive as a single file
IDE_FORMAT ($00B2)	Format a partition
IDE_PARTITION_ERASE ($00BE)	Delete a partition
IDE_PARTITION_RENAME ($00C1)	Rename a partition
IDE_PARTITION_WRITE ($00C7)	Write a partition entry
IDE_PARTITION_WINFO ($00CA)	Write type-specific partition information
IDE_PARTITION_GETINFO ($00D3)	Get byte from type-specific partition information
IDE_PARTITION_SETINFO ($00D6)	Set byte in type-specific partition information
IDE_DOS_UNPERMANENT ($00FA)	Remove permanent drive mapping
IDE_IDENTIFY ($01a2)	Return IDE drive identity information

Updated calls

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

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

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

DOS_OPEN
0106h (262)

Create and/or open a file

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

Open action

0. Error - File already exists.

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

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

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

4. Erase existing version. Follow create action.

Create action

0. Error - File does not exist.

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

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

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

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

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

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

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

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

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

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

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

(xxx = doesn't matter)

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

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

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

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

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

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

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

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

DOS_CATALOG
011Eh (286)

Fills a buffer with part of the directory.

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

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

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

Buffer format:

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

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

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

Entry format (13 bytes long):

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

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

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

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

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

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

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

DOS_FREE_SPACE
0121h (289)

How much free space is there on this drive?

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

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

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

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

DOS_GET_POSITION
0133h (307)

Get the file pointer.

```
ENTRY CONDITIONS
        B = File number

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

DOS_GET_EOF
0139h (313)

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

Does not affect the file pointer.

Does not consider soft-EOF.

```
ENTRY CONDITIONS
        B = File number

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

IDE_DOS_MAP ($00F1)

Map a drive to the specified partition or physical device

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

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

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

IDE_DOS_UNMAP ($00F4)

Remove mapping from the specified drive

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

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

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

IDE_SNAPLOAD ($00FD)

Load a snapshot

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

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

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

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

IDE_PATH ($01b1)

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

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

into which the returned path will be written

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

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

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

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

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

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

New calls

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

IDE_CAPACITY ($01b4)

Get card capacity

IN: C=unit (0 or 1)

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

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

IDE_GET_LFN ($01b7)

Obtain a long filename and other file information

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

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

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

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

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

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

IDE_BROWSER ($01ba)

Run the file browser

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

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

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

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

NOTES:

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

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

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

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

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

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

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

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

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

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

IDE_BANK ($01bd)

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

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

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

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

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

NOTE:

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

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

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

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

IDE_BASIC ($01c0)

Execute a BASIC command line

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

OUT(s):  Fc=1
```

System variable ERR_NR contains generated BASIC error code-1
($ff means BASIC command completed successfully)

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

NOTES:

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

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

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

IDE_WINDOW_LINEIN ($01c3)

Input line from current window stream

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

OUT: E=number of characters returned in input buffer

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

NOTES:

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

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

+3 BASIC errors may be invoked

IDE_WINDOW_STRING ($01c6)

Output string to current window stream

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

```
OUT: -

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

NOTES:

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

+3 BASIC errors may be invoked
```

IDE_INTEGER_VAR ($01c9)

```
Get or set NextBASIC integer variable

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

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

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

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

NOTE:

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

IDE_RTC ($01cc)

```
Query the real-time-clock module

IN:  -

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

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

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

NOTE:

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

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