Nextor 2.0 Alpha 1 Programmers Reference

By Konamiman, 7/2011

Index

1. Introduction 2
2. Changes in existing function calls 2
   2.1. _STROUT (09h) 2
   2.2. _ALLOC (1Bh) 2
   2.3. _RDABS (2Fh) and _WRABS (30h) 3
   2.4. _DPARM (31h) 3
   2.5. _DEFER (64h) 3
   2.6. _FORMAT (67h) 4
   2.7. _DOSVER (6Fh) 4
3. New function calls 6
   3.1. Get/set fast STROUT mode (_FOUT, 71h) 6
   3.2. Print a zero-terminated string (_ZSTROUT, 72h) 6
   3.3. Read absolute sectors from drive (_RDDRV, 73h) 6
   3.4. Write absolute sectors to drive (_WRDRV, 74h) 7
   3.5. Get/set reduced allocation information mode vector (_RALLOC, 75h) 7
   3.6. Get drive space information (_DSPACE, 76h) 7
   3.7. Lock/unlock a drive, or get lock state for a drive (_LOCK, 77h) 8
   3.8. Get information about a device driver (_GDRVR, 78h) 9
   3.9. Get information about a drive letter (_GDLI, 79h) 10
   3.10. Get information about a device partition (_GPART, 7Ah) 11
   3.11. Call a routine in a device driver (_CDRVR, 7Bh) 12
   3.12. Map a drive letter to a driver and device (_MAPDRV, 7Ch) 13
   3.13. Enable or disable the Z80 access mode for a driver (_Z80MODE, 7Dh) 14
4. New error codes 15
5. Extended mapper support routines 16
   5.1. BLK_ALLOC: Allocate a memory block 16
   5.2. BLK_FREE: Free a memory block 16
6. Other features 17
   6.1. Correction of the ESC-Y escape sequence bug in STROUT function 17
7. Contact 17

1. Introduction

Nextor is an enhanced version of MSX-DOS 2, the disk operating system for MSX computers. It is based on MSX-DOS 2.31, with which it is 100% compatible.

This document provides a reference of the new features that Nextor adds to MSX-DOS 2 from a developer point of view (basically the new function calls provided, but also some other useful information). The development of device drivers for Nextor is not covered in this document; this topic has a separate document devoted to itself, *Nextor 2.0 Alpha 1 Driver Development Guide*.

The reader of this document is assumed to have experience developing applications for MSX in general and for MSX-DOS 2 in particular (specifically, the information covered by chapter 3 of *MSX2 Technical Handbook* and the *MSX-DOS 2 Program Interface Specification* and *MSX-DOS 2 Function Codes Specification* documents is assumed to be known). Also, it is a good idea to get acquainted with Nextor by reading *Nextor 2.0 Alpha 1 User Manual* prior to this document.

WARNING: Nextor is in alpha state. This means that it has not been thoroughly tested, so please backup your valuable data before using it. Also, not all the planned features are implemented yet.

2. Changes in existing function calls

This section details what Nextor has changed in the function calls already existing in MSX-DOS 2. All the changes are non-breaking except a minor issue with 16 bit sector numbers in function _DPARM.

Only the changes introduced in Nextor are explained, the complete description of the functions is not provided here. See the *MSX-DOS 2 Function Codes Specification* document for more details on these function calls.

2.1. _STROUT (09h)

When the fast STROUT mode is enabled, the maximum printable string length is 511 characters; if the string is longer, only the first 511 characters will be printed. The fast STROUT mode is disabled by default, it must be explicitly enabled by using the new _FOUT function.

2.2. _ALLOC (1Bh)

When the reduced allocation information mode is enabled for a drive, this function will return a false total and/or free cluster count when necessary so that when multiplied by the sectors per cluster amount, the result will give 32MB or less when called for that drive. The reduced allocation information mode is disabled by default for all drives, it must be explicitly enabled for each desired drive by using the new _RALLOC function.

2.3. _RDABS (2Fh) and _WRABS (30h)

These functions will work only when the accessed drive contains a FAT12 filesystem. They will return a "Not a DOS disk" error when a drive containing a FAT16 filesystem or an unknown filesystem is accessed.

Strictly speaking, this is not a change from the behavior in MSX-DOS, since FAT12 was the only filesystem supported by that operating system. However, when a drive containing a FAT16 filesystem is read in MSX-DOS, its boot sector is mistook for a FAT12 boot sector, and the functions succeed. In Nextor FAT16 filesystems are explicitly rejected by these functions; this is done on purpose to prevent programs, such as CHKDSK or IMPROVE, that do low-level processing on drives assuming that they have a FAT12 filesystem, to cause data corruption when accidentally ran on these drives.

New applications should use instead the new functions _RDDRV and _WRDRV, which accept 32 bit sector numbers and allow access to any drive regardless of the contained filesystem.

2.4. _DPARM (31h)

This function now returns the total number of logical sectors as a 32 bit value at position +24..27 in the returned parameter block. Moreover, when this number is greater than 65535, the 16 bit sector count returned at position +9,10 will be zero.

Also, position +28 of the returned parameter block contains the filesystem type:

0: FAT12
1: FAT16
255: Other

Remember that Nextor can currently handle FAT12 and FAT16 filesystems only.

2.5. _DEFER (64h)

The parameters passed to the user routine in case of disk error are extended to support 32 bit sector numbers. In MSX-DOS 2, part of these parameters were as follows:

C:b3 - set if sector number is valid
DE = Sector number (if b3 of C is set)

In Nextor, these parameters are:

C:b3 - set if sector number is valid and fits in 16 bits (that is, b4 is set and HL=0)
C:b4 - set if sector number is valid
HL:DE = Sector number (if b4 of C is set)

2.6. _FORMAT (67h)

In MSX-DOS 2 this function accepts two special choice parameters, FFh and FEh, that do not actually format the disk but generate a MSX-DOS 2 boot sector, including the disk parameters, based on the media ID of the disk. This feature is used by the FIXDISK program to convert old MSX-DOS 1 disks into MSX-DOS 2 disks.

Nextor adds three new choice parameters:

- FDh: Assuming that the disk has a FAT12 or FAT16 filesystem with valid disk parameters on the boot sector (otherwise a "Not a DOS disk" will be returned), a standard boot sector will be composed for the disk by (1) setting the manufacturer name as "NEXTOR", and (2) generating an extended block (byte 29h, plus volume ID, plus volume name, plus "FAT12" or "FAT16" mark). The disk parameters will not be modified. If the disk contains an extended block already, only the manufacturer name will be changed (thus maintaining the existing volume ID). This choice is useful for using the dirty disk flag feature on disks already formatted by another system.

- FCh: It is the same as FDh, but if the disk filesystem is FAT12, a MSX-DOS 2 boot sector (with the "VOL_ID" string) will be composed instead of a standard boot sector. If the disk filesystem is FAT16, this choice works the same way as FDh. This choice is useful if the disk is to be used on MSX-DOS 2 systems.

- FBh: Will perform a "quick format" on the disk, by simply clearing the FAT and root directory areas. As with the other two new choices, the disk must have a valid FAT12 or FAT16 boot sector, otherwise a "Not a DOS disk" error will be returned.

When the disk is actually formatted (choice 1..9), a MSX-DOS 2 boot sector will always be generated.

2.7. _DOSVER (6Fh)

This function call has been expanded in order to allow applications to detect whether they are running MSX-DOS 2 or Nextor, while at the same time still working for applications that expect the operating system to be MSX-DOS.

The procedure for detecting Nextor is as follows. The application must invoke the _DOSVER function with the following “magic numbers” as input parameters:

B = 5Ah
HL = 1234h
DE = ABCDh
IX = 0

When the operating system is MSX-DOS 2, then this function call will return IX = 0 (since MSX-DOS does not modify the IX and IY registers unless they return a result, and no result is returned in these registers by this function). When running Nextor and the magic numbers are NOT supplied, the function call will return IX and IY unmodified as well, to maintain compatibility with MSX-DOS 2.

When the operating is Nextor and the magic numbers are provided, then the following information is returned:

B = Emulated MSX-DOS kernel major version number (always 2)
C = Emulated MSX-DOS kernel minor version number (always 31h)
D = NEXTOR.SYS major version number in BCD format
E = NEXTOR.SYS minor version number in BCD format
HL = Address of an operating system descriptive string in kernel ROM
IXh = 1
IXl = Nextor major version number (2-15)
IYh = Nextor secondary version number (0-15)
IYl = Nextor revision number (0-255)

The procedure for detecting the operating system for Nextor aware applications is then as follows:

1. Call the DOSVER function with the magic numbers set.
2. If there is an error (A<>0) then the operating system it is neither MSX-DOS nor Nextor.
3. If B<2 then the operating system is MSX-DOS 1.
4. If IX is 0 then the operating system is MSX-DOS 2. Look at registers B and C for the version number.
5. If IXh is 1 then the operating system is Nextor. Look at registers IXh, IYh and IYl for the version number.
6. If IXh is neither 0 nor 1 then the operating system is neither MSX-DOS nor Nextor.

When running in MSX-DOS 1 mode, at this time there is no way to distinguish whether Nextor or an original MSX-DOS 1 kernel is being ran. This will change in a future version of Nextor.

The value returned in HL is a pointer to a zero-terminated printable string that describes the operating system running, for example "Nextor kernel version 2.0 alpha 1". The string resides in the kernel master slot (slot number is available at 0F348h) and can be readed via standard RDSLT calls.

3. New function calls

This section details the new function calls introduced by Nextor. These are invoked the same way as the existing MSX-DOS calls, by setting the function number in register C and calling address 0005h or F37Dh. The specified short name for each function (for example "_FOUT") is the suggested name for referring the function call in code, and is also the name used for function cross references in this manual.

3.1. Get/set fast STROUT mode (_FOUT, 71h)

Parameters: C = 71H (_FOUT)
A = 00H => get fast STROUT mode
01H => set fast STROUT mode
B = 00H => disable (only if A=01H)
FFH => enable (only if A=01H)

Results: A = Error
B = Current fast STROUT mode

This function enables or disables the fast STROUT mode. When enabled, the _STROUT and _ZSTROUT functions will work faster, but the maximum printable string length will be 511 characters; if the string is longer, only the first 511 characters will be printed.

3.2. Print a zero-terminated string (_ZSTROUT, 72h)

Parameters: C = 72H (_ZSTROUT)
DE = Address of string

Results: A = 0 (never returns an error)

Prints on the screen the string pointed by DE, the string must be terminated with a zero character. This function is affected by the fast STROUT mode.

3.3. Read absolute sectors from drive (_RDDRV, 73h)

Parameters: C = 73H (_RDDRV)
A = Drive number (0=A: etc.)
B = Number of sectors to read
HL:DE = Sector number

Results: A = Error code (0=> no error)

This function reads sectors directly from a drive. Unlike _RDABS, this function is able to read sectors regardless of the filesystem viewed through the drive (FAT12, FAT16 or an unknown filesystem), and even when there is no filesystem at all.

The sectors will be read to the current disk transfer address. Any disk error will be reported by the system in the usual way.

3.4. Write absolute sectors to drive (_WRDRV, 74h)

Parameters: C = 74H (_WRDRV)
A = Drive number (0=A: etc.)
B = Number of sectors to write
HL:DE = Sector number

Results: A = Error code (0=> no error)

This function writes sectors directly to a drive. Unlike _WRABS, this function is able to write sectors regardless of the filesystem viewed through the drive (FAT12, FAT16 or an unknown filesystem), and even when there is no filesystem at all.

The sectors will be written from the current disk transfer address. Any disk error will be reported by the system in the usual way.

3.5. Get/set reduced allocation information mode vector (_RALLOC, 75h)

Parameters: C = 75H (_RALLOC)
A = 00H => get current vector
01H => set vector
HL = new vector (only if A=01H)

Results: A = 0 (never returns an error)
HL = Current vector

This function obtains or sets the reduced allocation information mode vector. The vector assigns one bit for each drive; bit 0 of L is for A:, bit 1 of L is for B:, etc. This bit is 1 if the reduced allocation mode is currently enabled (when getting vector) or to be enabled (when setting vector) for the drive, 0 when the mode is disabled or to be disabled.

3.6. Get drive space information (_DSPACE, 76h)

Parameters: C = 76H (_DSPACE)
E = drive number (0 = default, 1 = A:, etc)
A = 00H => get free space
01H => get total space

Results: A = error code
HL:DE = space in kilobytes
BC = extra space in bytes

This function returns the total or free space for a drive. The space information is always returned in Kilobytes, regardless of the type and the cluster size of the filesystem mapped to the drive.

The "extra free space in bytes" result will be different from zero only when the minimum allocation unit of the drive is not a whole number of kilobytes. In case of FAT drives, it will be non-zero (specifically, it will be 512) only when the drive uses one sector per cluster and the cluster count is odd. For example, for a drive having one sector per cluster and 15 free clusters, this function will return HL=0, DE=7 and BC=512 when called with A=0 for that drive.

The space information returned by this function is always real, it is not affected by the reduced allocation information mode.

3.7. Lock/unlock a drive, or get lock state for a drive (_LOCK, 77h)

Parameters: C = 77H (_LOCK)
E = physical drive (0=A:, 1=B:, etc)
A = 00H => get lock status
01H => set lock status
B = 00H => unlock drive (only if A=01H)
FFH => lock drive (only if A=01H)

Results: A = Error code
B = Current lock status, same as input

This function locks or unlocks a drive, or gets the current lock state for a drive. When a drive is locked, Nextor will assume that the media on that drive will never be changed, and therefore will never ask the associated driver for media change status; thus resulting in an overall increase on media access speed. This is useful when using removable devices, such as multimedia cards, as the main storage device.

In order to be locked, the drive must be mapped to a valid filesystem (that is, the drive must be accessible); otherwise an error will be returned and the drive will not be locked.

Once a drive is locked, any disk error that is aborted will automatically unlock the drive.

Locking and unlocking operations cause all the buffers for the drive to be flushed and invalidated. Also, cached disk parameters for the media are deleted so the next access to the media will re-read them.

It is possible to lock non-removable devices associated to device-based drivers, however it makes no sense to do that, since Nextor will never ask for media change status for this kind of devices.

Locking feature must be used with care. Changing the device of a locked drive without first unlocking it may result in data corruption, both in the original (locked) media and in the newly inserted media.

3.8. Get information about a device driver (_GDRVR, 78h)

Parameters: C = 78H (_GDRVR)
A = Driver index, or 0 to specify slot and segment
D = Driver slot number (only if A=0)
E = Driver segment number, FFh for drivers in ROM
(only if A=0)
HL = Pointer to 64 byte data buffer

Results: A = Error code
HL = Filled with data about driver

This function returns information about a device driver present in the system.

The device driver can be specified by index or by slot and segment number pair. To specify the driver by index, set the index number (starting at 1) in register A; the slot and segment number for the driver is returned in the data buffer together with other driver information. This is useful to discover which drivers are present in the system.

If you know already the slot and segment numbers of the driver you want to gather information about, set these parameters in registers D and E, and set A=0. The slot and segment numbers are returned anyway in the data buffer, as in the case of specifying a driver index.

An .IDRVR error will be returned if there is no driver associated to the specified index, or if there is no driver with the specified slot and segment pair. There is no way to know in advance how many drivers are present in the system, so to discover all the drivers this function must be invoked several times, starting with driver index 1 and increasing the index number until a .IDRVR error is obtained.

The information returned in the data buffer is as follows:

+0: Driver slot number
+1: Driver segment number, FFh if the driver is embedded within a Nextor or MSX-DOS kernel ROM
(always FFh in Alpha 1)
+2: Number of drive letters assigned to this driver at boot time
+3: First drive letter assigned to this driver at boot time (A:=0, etc),
unused if no drives are assigned at boot time
+4: Driver flags:
bit 7: 1 => the driver is a Nextor driver
0 => the driver is a MSX-DOS driver
(embedded within a MSX-DOS kernel ROM)
bits 6-1: Unused, always zero
bit 0: 1 => the driver is a device-based driver
0 => the driver is a drive-based driver
+5: Driver main version number
+6: Driver secondary version number
+7: Driver revision number
+8: Driver name, left justified, padded with spaces (32 bytes)
+40..+63: Reserved (currently always zero)

In the case of MSX-DOS drivers, the driver flags byte is always zero, and no information about driver version number or driver name is returned.

3.9. Get information about a drive letter (_GDLI, 79h)

Parameters: C = 79H (_GDLI)
A = physical drive (0=A:, 1=B:, etc)
HL = Pointer to 64 byte data buffer

Results: A = Error code
HL = Filled with data about the drive

This function return information about a given drive letter. The information returned in the data buffer is as follows:

+0: Drive status
  0: Unassigned
  1: Assigned to a storage device attached to a Nextor or MSX-DOS driver
  2: Unused in Alpha 1
  3: Unused in Alpha 1
  4: Assigned to the RAM disk (all other fields will be zero)
+1: Driver slot number
+2: Driver segment number, FFh if the driver is embedded within a Nextor or MSX-DOS kernel ROM (always FFh in Alpha 1)
+3: Relative drive number within the driver (for drive-based drivers only; FFh if device-based driver)
+4: Device index (for device-based drivers only; 0 for drive-based drivers and MSX-DOS drivers)
+5: Logical unit index (for device-based drivers only; 0 for drive-based drivers and MSX-DOS drivers)
+5..+8: First device sector number (for devices in device-based drivers only; always zero for drive-based drivers and MSX-DOS drivers)
+9..+63: Reserved (currently always zero)

If a drive larger than the maximum drive number supported by the system is specified, an .IDRV error will be returned. Note that if a drive number is specified which is legal in Nextor, but is currently not assigned to any driver, then no error will be returned, but an empty information block will be returned (the drive status byte should be checked).

The "first device sector number" is the absolute device sector number that is treated as the first logical sector for the drive; usually it is either the starting sector of a device partition, or the device absolute sector zero, if the device has no partitions. Note that you can't test this value against zero to check whether the drive is assigned to a block device on a device-based driver or not (use the “drive status” field for this purpose).

3.10. Get information about a device partition (_GPART, 7Ah)

Parameters: C = 7AH (_GPART)
A = Driver slot number
B = Driver segment number, FFh for drivers in ROM
(must be always FFh in Alpha 1)
D = Device index
E = Logical unit index
H = Primary partition number (1 to 4)
L = Extended partition number (0 for an entry in the primary partition table)

Results: A = Error code
B = Partition type code, 0 if the specified partition does not exist
HL:DE = Starting device absolute sector number of the partition

This function returns information about a device partition. It only works on device-based drivers; if a non-existing driver, a drive-based driver, or a MSX-DOS driver is specified in A and B, then an .IDRVR error will be returned. If the specified device and/or logical unit do not exist in the driver, an .IDEVL error will be returned.

Storage devices are usually divided in partitions, each one being an independent logical volume residing in a contiguous block of sectors in the media. This function allows finding the starting sector of a given partition in the media, usually in order to map it to a drive letter by using the MAPDRV function, so that the contained filesystem can be accessed by Nextor.

The partition type code returns information about the filesystem that the partition holds. The code may be one of the following:

0: None (the partition with the specified number does not exist)
1: FAT12
4: FAT16, smaller than 32MB (obsolete)
5: Extended (see below)
6: FAT16

There are many more partition type codes defined, but they refer to filesystems that can't be handled by Nextor so they are not listed here.

A device can have up to four primary partitions, numbered 1 to 4. In order to accommodate more than four partitions, partition number 2 may be of a special type named "Extended". An extended partition is actually a container for more partitions; there is no limit in the number of extra partitions that a partition of type "Extended" can contain. Primary partitions 3 and 4 do not exist when partition 2 is extended.

In order to enumerate all the partitions existing in a device , the following procedure should be followed, to take in account the possible presence of extended partitions:

1. Search partition 1-0 (primary number 1, extended number 0).

2. Search partition 2-0. If it exists and is of type "Extended", search partitions 2-1, 2-2, 2-3, etc, until a partition code 0 is returned.

3. If partition 2-0 does not exist or is not of type "Extended", search partitions 3-0 and 3-4.

Note that it is possible that a device has no partitions at all. In this case, it is still possible that the device contains a valid filesystem, mapped to the absolute device sector zero; this is indeed the case of floppy disks and devices with very small capacity.

When a partition is mapped to a drive letter, the partition first sector will always be examined in order to determine the actual filesystem holded by the partition. Nextor will never rely in the partition type code to determine the filesystem type.

Nextor needs to read the device in order to search for partitions. If there is any error when accessing the device (for example, not ready), an error code will be returned. The standard system error handling routine (or the user error handling routine, if one is defined with _DEFER) will NOT be invoked.

When the specified partition does not exist in the device (for example, when a primary partition number larger than 4 is specified, or when an extended partition number is specified for a non-extended primary partition), then B=0 and A=.IPART will be returned.

3.11. Call a routine in a device driver (_CDRVR, 7Bh)

Parameters: C = 7BH (_CDRVR)
A = Driver slot number
B = Driver segment number, FFh for drivers in ROM
(must be always FFh in Alpha 1)
DE = Routine address
HL = Address of a 8 byte buffer with the input register values for the routine

Results: A = Error code
BC, DE, HL = Results from the routine
IX = Value of AF returned by the routine

This function allows direct invocation of a routine in a device driver. Routines for any driver type (MSX-DOS, device-based and drive-based) can be invoked with this function, however it is intended primarily for device-based drivers, in order to enumerate devices (DEV_INFO and LUN_INFO) and directly access the device absolute sectors (DEV_RW routine), for example to develop device partitioning tools. The available routines for device drivers are enumerated and described in detail in the *Nextor 2.0 Alpha 1 Driver Development Guide* document.

The input value of registers AF, BC, DE and HL for the routine must be provided in an 8 byte buffer pointed by HL. The order of the register values in the buffer is as follows: F, A, C, B, E, D, L, H. The output values of these registers, on the other hand, are returned directly in the registers themselves; except the output value of AF which is returned in IX.

Some routines accept data from, or write data to, memory buffers supplied by the user. There are two limitations for exchanging data with the driver routines in this way: first, the buffer must be in the primary mapper slot; and second, the buffer may not be partially or totally in page-1. These limitations do not apply for the 8 byte register buffer (remember however that when invoking Nextor function calls via the F37Dh hook, no parameters can be passed in page-1). The register buffer is only used before effectively executing the driver routine, therefore there is no problem if it overlaps with any buffer used by the routine to return data.

An .IDRVR error will be returned by this function call if a non-existing driver is supplied in A and B. Use the Nextor function _GDRVR to discover the location of the existing drivers.

3.12. Map a drive letter to a driver and device (_MAPDRV, 7Ch)

Parameters: C = 7CH (_MAPDRV)
A = physical drive (0=A:, 1=B:, etc)
B = Action to perform
0: Unmap the drive
1: Map the drive to its default state
2: Map the drive by using specific mapping data
HL = Address of a 8 byte buffer with mapping data (if B=2)
Results: A = Error code

This function allows mapping a drive number to a specific combination of device number, logical unit number, and starting absolute device sector number, within a device-based driver. It also allows to revert back the drive mapping to its default state (the state at boot time), and to completely unmap the drive.

If B=0 at input, the drive will be unmapped. This means that the drive will be unavailable from that moment, and any attempt to access it will result in an "Invalid drive" error. If the drive is already unmapped, nothing will happen and no error will be returned.

If B=1 at input, the drive will be reverted to its default state. If at boot time the drive was unmapped (not assigned to any driver), or was mapped to a drive on a MSX-DOS driver or on a drive-based driver, then the drive will be reverted to the same state. If at boot time the drive was assigned to a device-based driver, then an auto-assign procedure will be performed for this drive, using the same rules as the automatic mapping procedure performed at boot time (the automatic mapping procedure is described in the *Nextor 2.0 Alpha 1 User Manual* document). This may result or not on the drive having the same mapping as it had at boot time, depending on the presence of removable devices in the associated driver and the state of the other drives.

If the automatic mapping procedure resulting from invoking this function with B=1 fails (because there are no suitable devices or partitions), the drive will be unmapped, regardless of its previous mapping state. An .IDEVL error will be returned in this case.

If B=2 at input, the drive will be mapped according to the mapping data provided in the buffer pointer by HL. It is possible to map any system drive to any device-based driver by using this method, even drives that were unmapped or were mapped to a different driver at boot time. The contents of the mapping data buffer must be as follows:

+0: Driver slot number
+1: Driver segment number, FFh if the driver is embedded within a Nextor kernel ROM
(must be always FFh in Alpha 1)
+2: Device number
+3: Logical unit number
+4..+7: Starting sector

An .IDRVR error will be returned if the specified driver does not exist or is not a device-based driver. An .IDEVL error will be returned if the device with the specified device and logical unit numbers does not exists in the driver. In these cases, the previous drive mapping will not be modified.

A .RAMDX error will be returned if the drive specified is H: and a RAM disk exists.

It is not possible to assign the same combination of driver, device, logical unit, and starting sector to more than one drive at the same time; this is to prevent data corruption. If the mapping data provided matches the mapping data of other drive, then a .IPART error will be returned.

The "starting sector" parameter is the device absolute sector number that will be used as the sector zero for the drive. Usually this will be the starting sector of a device partition, obtained via a call to the _GPART function. Note however that no checking is done for the presence of an actual (and recognized by Nextor) filesystem starting in the specified sector; if no valid filesystem is found, the _MAPDRV function will succeed, but the next access to the drive will return a "Not a DOS disk" error.

Also, it is possible to map a drive to a removable device which has no media inserted. In this case, the _MAPDRV function will succeed, but the next access to the drive will return a "Disk offline" error.

It is not possible to map two drives to the same combination of driver, device, logical unit, and start sector; this is to prevent data corruption resulting from dealing with unsynchronized sector buffers. An .IDEVL error will be returned in this case. In order to change the drive letter for a given mapping, the old drive letter must be first unmapped.

Also, note that it is not possible to explicitly map a drive to a MSX-DOS driver or a drive-based driver.

Before changing the mapping state of a drive, any open file handles relative to that drive will be closed by this function. This is equivalent to invoking the CLOSE function call for each of these file handles, so disk errors may arise if there are dirty buffers for that drive and there are errors when flushing them to the device.

3.13. Enable or disable the Z80 access mode for a driver (_Z80MODE, 7Dh)

Parameters: C = 7DH (_Z80MODE)
A = Driver slot number
B = 00H => get current Z80 access mode
01H => set Z80 access mode
D = 00H => disable Z80 access mode (only if A=01H)
FFH => enable Z80 access mode (only if A=01H)

Results: A = Error code
D = Current Z80 access mode for the specified driver, same as input

This function works on MSX Turbo-R computers only. On MSX1/2/2+ it will always return an .IDRVR error.

This function allows enabling or disabling the Z80 access mode for a MSX-DOS driver (a driver embedded within a MSX-DOS kernel ROM). When the Z80 access mode for a driver is enabled, Nextor will switch to the Z80 CPU prior to accessing any drive associated to the driver. When the Z80 access mode for a driver is disabled, no CPU switching is performed, being therefore possible to access the drives on the driver in R800 mode.

When the computer boots, the Z80 access mode is enabled for all MSX-DOS drivers. This is necessary because some old device drivers do not work when accessed in R800 mode, and it would not be possible to boot from a device on one of these drivers. Once the system boot has finished, this function can be used to disable (and later re-enable if necessary) the Z80 access mode for any MSX-DOS driver.

Note that the Z80 access mode is enabled or disabled for a whole driver, affecting all the drives associated to the driver. It is not possible to selectively enable or disable the Z80 access mode for individual drives.

The Z80 access mode applies to MSX-DOS drivers only. Nextor will never change the CPU when accessing drive-based and device-based drivers.

4. New error codes

New error codes are defined to handle error conditions when managing the new features of Nextor. These errors are the following:

- Invalid device driver (.IDRVR, 0B6h)

An operation involving a device driver has been requested but the specified driver does not exist, or is not of the valid type (for example, the driver is a MSX-DOS or drive-based driver but a device-based driver is required).

- Invalid device or LUN (.IDEVL, 0B5h)

An operation involving a device on a device-based driver has been requested but the specified device does not exist in the driver, or the specified logical unit does not exist in the specified device.

- Invalid partition number (.IPART, 0B4h)

Information about a disk partition on a device has been requested, but the specified partition does not exist in the device.

- Partition is already in use (.PUSED, 0B3h)

An attempt has made to map a drive to a driver, device and starting sector number; but there is already another drive which is mapped to the same combination of driver, device, logical unit, and starting sector number.

5. Extended mapper support routines

The original MSX-DOS 2 mapper support routines have been extended with two new functions that allow the allocation of small blocks of memory (from 1 to 16378 bytes) within an allocated or TPA segment. Entries for these functions are available as an extension of the mapper support routines jump table whose address can be obtained by using extended BIOS. The names and locations in the jump table of these new routines are:

+30h: BLK_ALLOC
+33h: BLK_FREE

Both routines work on the memory that is switched on page 2 at the moment of calling them. It may be an explicitly allocated segment, a TPA segment, or even non-mapped RAM: they will work on any writable memory that is visible on page 2. However a segment will be assumed to be switched on page 2 for documentation purposes.

Following is the description of these routines.

5.1. BLK_ALLOC: Allocate a memory block

Entry: HL = Required size (1 to 16378 bytes)
Returns: On success:
HL = Address of the allocated block (always a page 2 address)
A = 0 and Z set
On error (not enough free space on segment):
HL = 0
A = .NORAM and Z reset

This routine tries to allocate a memory block of the specified size on the segment currently switched on page 2, and returns the address of the allocated block if it succeeds, or a "Not enough memory" error if not. The segment must have been previously initialized by calling the BLK_FREE routine with HL=0, otherwise the result is unpredictable.

5.2. BLK_FREE: Free a memory block

Entry: HL = Address of the allocated block as returned by BLK_ALLOC,
or 0 to initialize the segment

This routine frees a memory block on the segment currently switched on page 2. The specified address must be a block address previously returned by the BLK_ALLOC routine on the same segment, otherwise the result is unpredictable. The freed space will become available for new allocations.

All the state information about allocated and free blocks is stored on the segment itself, Nextor does not store any internal information about block memory allocation. This means that when all the allocated blocks on a given segment are no longer needed, it is not needed to explicitly free all blocks one by one; instead, the segment may be overwritten with any other data, the segment itself may be freed, or (in case of TPA segments) application may terminate directly.

When called with HL=0, this routine initializes the segment currently switched on page 2 for block memory allocation. It is necessary to do this once before performing any block allocation on the segment. Also, this is useful on segments that already have allocated blocks, as a fast way to free all blocks at once.

6. Other features

This section describes other miscellaneous new features offered by Nextor.

6.1. Correction of the ESC-Y escape sequence bug in STROUT function

The STROUT function prints a string finished with a "$" character. There is an escape sequence that allows positioning cursor at any location in the screen, the sequence is: ESC (27) "Y" (89) x+32 y+32, where (x,y) is the desired character position.

The bug appears when this escape sequence is used and either the x or the y coordinate are 4. In this case, the third or fourth byte of the sequence becomes 36, which is the ASCII code of "$"; then MSX-DOS incorrectly assumes that this is the end of string mark and the string is truncated.

This bug is corrected in Nextor, so the ESC-Y escape sequence can be safely used.

7. Contact

You can get the latest version of Nextor and the associated tools at Konamiman's MSX page:

http://www.konamiman.com

For bug reports or suggestions, please write at:

konamiman@konamiman.com