CONFIDENTIAL - INTERNAL CIRCULATION ONLY

Document PER-3
Title ..... EXDOS - Unit Handler Specification
Issue 2
Date ..... 23rd April 1985

CONTENTS

Note: This document describes the general interface between EXDOS and unit handlers to enable a programmer to write an additional unit handler and link it in to EXDOS. Specific implementation details of the built in unit handler (UNITH) are contained in the document PER-4. Related documents are:

PER-1	EXDOS - System Overview
PER-2	EXDOS - DISKIO Specification
PER-4	DISKIO and UNITH Implementation Notes
PER-5	EXDOS - System Specification
PER-16	IS-DOS - System Specification

1. INTRODUCTION

A unit handler is a section of code which provides the interface between the EXDOS filing system handler (FISH), and a particular piece of disk hardware. EXDOS comes with one unit handler (called UNITH) already contained in the ROM and automatically linked in. This provides EXDOS with an interface to up to four disk drives connected to the Enterprise disk controller card.

Additional unit handlers can be linked in to EXDOS either from ROM modules or from RAM based system extensions at any time, and EXDOS will automatically search for ROM based ones when it starts up. EXDOS actually comes with an additional unit handler which may be linked in to provide a RAM-DISK facility. External unit handlers could be linked in to support more floppy disks, hard disk drives or any other random access storage medium such as non-volatile RAM.

Each logical disk drive is called a unit, corresponding to the idea of unit numbers in EXOS. EXDOS allocates unit numbers to unit handlers in sequence from 1...26. Units 1...4 are always allocated to UNITH and other numbers are allocated to extension unit handlers as they are linked in. A single unit handler can support any number of units, each of which may correspond to a separate disk drive or just to a separate partition of a disk.

The level of interface provided by EXDOS unit handlers is very similar to that provided by block devices in MS-DOS 2.00, although there are some complexities due to the paged nature of the Enterprise's memory.

2. LINKING IN UNIT HANDLERS

Before EXDOS can use a unit handler it must be linked in. This is done automatically for UNITH but any other unit handler must be explicitly linked in. There are two linking procedures, one for unit handlers in extension ROMs, and the other for unit handlers in RAM which are linked in later. These two procedures are described below. UNITH is rather a special case and is covered in more detail in the implementation notes (PER-4).

2.1 EXTENSION ROM UNIT HANDLERS

Unit handlers in extension ROMs are linked in when the machine is switched on. EXOS will call all extension ROMs to initialise them (action code 8), including the EXDOS ROM and any ROMs containing extension unit handlers. When EXDOS is called it will link in UNITH and then pass the string "EXDOS UNIT" around all other ROMs.

Any ROM with an extension unit handler in should detect this string and return certain parameters (see below) to EXDOS to allow it to link in this unit handler. EXDOS will link it in and then repeat the string scan call to get the next extension. Once an extension has been linked in it should ignore this string to allow it to pass on to other extensions. EXDOS will stop the scan when it gets no response to its string.

Having linked all unit handlers in, EXDOS will pass a COLD INIT command to each one starting with UNITH. The function of this COLD INIT command is described in section 4. Note that since the order of initialisation of extension ROMs is arbitrary, a ROM with a unit handler in may get the "EXDOS UNIT" string either before or after EXOS has initialised the ROM. Since this could cause problems, it is advised that all extension unit handler ROMs ignore the EXOS initialisation call, and rely instead on the COLD INIT command from EXDOS which will always occur after the unit handler has been linked in.

The parameters returned by a unit handler in response to the string scan are:

```
          A = 0   (Status code)
          C = 0   (Action code - zero stops scan)
          B = Segment containing RAM area
         DE = Address of RAM area for unit descriptors
               ?? bytes per unit  (May be in any Z-80 page)
  (DE+0)    = Number of units supported (1...22)
(DE+1...2)  = Address of entry point to unit handler
                (Must be in Z-80 page-3)
  (DE+3)    = Segment containing unit handler code
```

The RAM area is used by EXDOS to store various internal data for each unit supported by this handler, it should not be accessed by the handler itself. The address in DE will passed back to the unit handler in register IY whenever it is called in future, with the segment in B' (like EXOS calling a device driver).

Typically, a unit handler ROM will allocate itself some page-2 RAM when EXOS offers it (action code 7), which occurs before any of the initialisation procedure described above. When the unit handler recieves the "EXDOS UNIT" string it will return in DE the address of its RAM area (passed to the ROM by EXOS in IY), and the system segment number (0FFh) in B. Then whenever the unit handler is called by EXDOS, it will have the same pointer in IY as if the ROM had been called by EXOS, making it simple to access any variables etc.

2.2 RAM RESIDENT UNIT HANDLERS

RAM resident unit handlers are linked in after EXDOS has been fully initialised. Typically they will be contained in system extensions loaded from disk. The unit handler must tell EXDOS of its presence by passing it a string using the "scan system extensions" EXOS call. This string contains the address of a RAM area for unit descriptors, initialised to contain the number of units and entry point address, just like the parameters returned by ROM unit handlers (see section 2.1).

The format of the EXOS string is defined by the following assembler source. Note that the address and segment are contained in the string as bytes, not encoded in ASCII, and these bytes must be included in the length. Thus the length of this string is 13 bytes.

```
db      "EXDOS UNIT"
dw      address             ;Address or RAM area
db      segment             ;Segment of RAM area
```

The start of the RAM area must be initilaised to the number of units supported and the entry point address of the unit handler, exactly as for ROM resident unit handlers (see section 2.1).

EXDOS will immediately pass a COLD INIT command to the unit handler and from then on it will be treated identically to a ROM resident unit handler. Note that this command will be received by the unit handler before EXDOS returns from the "scan system extensions" call. If the COLD INIT command returns an error code, then it will be passed back by EXDOS as the status code for the "scan extensions" call. If the COLD INIT command always returns zero then a non-zero error code from the "scan extensions" call means that EXDOS is not present.

Note that unit handlers which are contained in RAM resident system extensions (the normal case) should link themselves into EXDOS when they are first initialised by EXOS (action code 8). However they must not attempt to link themselves in again when the system extension is re-initialised by EXOS. This can be tested for by setting a flag somewhere in the code of the system extension.

3. UNIT HANDLER COMMANDS - GENERAL

Unit handlers are always called by EXDOS, there is no facility for a user program to call them directly. Each handler has a single entry point which it passed to EXDOS in the RAM area when it was linked in (see section 2).

Whenever the unit handler is called by EXDOS, the following parameters are passed in registers.

```
 B' = Segment containing RAM area
 IY = Address of start of RAM area
 IX = Disk transfer address
  A = Command code  -  0 = COLD INIT
                       1 = WARM INIT
                       2 = MEDIA CHECK
                       3 = BUILD UPB
                       4 = READ
                       5 = WRITE
                       6 = WRITE WITH VERIFY
                       7 = FORCE UPB
  C = Relative unit number (0...n-1)
```

The other registers are used for various purposes depending on the command (see below). The disk transfer address pointed to by IX will always be in Z-80 page-1, with the correct segment paged in.

All commands should return a standard EXOS status code in register A, and various results in other registers depending on the command (see below). Note that EXDOS defines certain additional error codes which will be useful for unit handlers to return (see section 7). Any of the registers (BC,DE,HL,IX,IY,AF',BC',DE',HL') which are not used for returning results may be corrupted since EXDOS saves them itself.

The entry point will always be called with the unit handler code paged into Z-80 page-3. Page-2 will contain the system segment (0FFh), page-1 will contain the segment to be for disk data transfer and page-0 will contain the page-zero segment.

Note that the segment containing the RAM area, passed in register B', is not explicitly paged in by EXDOS, since there are no spare pages. If the RAM area is in the system segment (which will normally be the case for ROM resident unit handlers), or the same segment as the code (usually for RAM resident handlers) then it can be accessed directly since these segments are paged in and IY is pointing to the RAM area. However if the RAM area is in some other segment then the unit handler will have to do some paging to access it, not forgetting the segment for data transfer.

The relative unit number passed in register C will be in the range (0....n-1) where "n" is the number of units supported by this device. Thus a given unit handler will always receive the same relative unit numbers, regardless of what other unit handlers are linked in, and it does not need to know which physical unit numbers it is supporting.

4. COLD and WARM INIT COMMANDS

The COLD INIT command will only be received once by each unit handler immediately after it is first linked in. A WARM INIT command will be received whenever EXOS does a re-initialisation of system extensions, which will normally be when the RESET button is pressed or when a new applications program starts up.

EXDOS does not actually require the unit handler to do anything in response to this command, it is just provided in case the unit handler has any internal initialisation to do. Typically the unit handler will just reset a few internal variables and re-initialise any hardware which it uses. The 512 byte scratch area pointed to by IX (in page-1) can be used for reading data from a disk if the unit handler feels that this will help it.

```
PARAMETERS:    B':IY = Segment and address of RAM area
                   A = 0 (COLD RESET)
                       1 (WARM RESET)
                  IX = Address of 512 byte scratch area
                   C = 0    (All units initialised at once)

   RESULTS:        A = Status code (normally 0)
```

5. MEDIA CHECK and BUILD UPB

These commands are provided to allow automatic logging of changed disks and automatic handling of different disk formats. For each unit, EXDOS maintains a current "Unit Parameter Block" (UPB) and "media descriptor byte".

5.1 UNIT PARAMETER BLOCK

The UPB is identical to the BPB (Bios Parameter Block) used by MS-DOS 2.00 and contains sufficient information to define the disk format. The last three entries (marked with a "*" in the table below are not used at all by EXDOS but are provided to help the unit handler understand the media. The format of the UPB is:

```
  0...1  -  Bytes per sector.  Must always be 512 for EXDOS.
     2   -  Sectors per cluster.  Must be a power of 2.
  3...4  -  Reserved sectors including boot sector.
     5   -  Number of FATs.
  6...7  -  Number of root directory entries.
  8...9  -  Total number of sectors on disk, including
            reserved sectors but excluding hidden sectors.
    10   -  Media descriptor byte.
 11..12  -  Number of sectors occupied by each FAT.

* 13..14  -  Sectors per track.
* 15..16  -  Number of heads.
* 17..18  -  Number of hidden sectors.
```

There is an optional extension to the UPB as specified below. It is not necessary for a unit handler to support this extension if it doesn't want to. If it is supported then EXDOS can utilise its full checking for disk change.

```
    19   -  Z-80 "RET" instruction for MSX-DOS compatibility

 20..52  -  33 bytes reserved for possible extensions to MS-
            DOS or MSX-DOS boot sectors.

 23..58  -  "VOL_ID" string.  This string can be tested for to
            determine if the optional extension is present.
            If this string is not here then the dirty disk
            flag and volumne id are not present.

    59   -  Dirty disk flag to allow un-deletion of files.

 60..63  -  4 byte random volume id.  This is a unique number
            put on by FORMAT or DISKCOPY to distinguish this
            disk from any other.  The four bytes are all in
            the range 0...127.
```


5.2 MEDIA DESCRIPTOR BYTE

The "media descriptor byte" is a single byte which has no meaning to EXDOS but should be different for all disk formats which the unit handler can cope with. Certain standard media bytes have been defined (for MS_DOS and MSX-DOS) and these should be adhered to:

Media descriptor	0F8h	0F9h	0FAh	0FBh	0FCh	0FDh	0FEh	0FFh
Sectors/Track	9	9	8	8	9	9	8	8
Number of tracks	80	80	80	80	40	40	40	40
Number of sides	1	2	1	2	1	2	1	2
Bytes/sector	- - - - - - - - - - -			512	- - - - - - - - - - -			
Sectors/cluster	2	2	2	2	1	2	1	2
Reserved sectors	- - - - - - - - - - -			1	- - - - - - - - - - -			
Number of FATs	- - - - - - - - - - -			2	- - - - - - - - - - -			
Directory entries	112	112	112	112	64	112	64	112
Total sectors	720	1440	640	1280	360	720	320	640
Sectors/FAT	2	3	1	2	2	2	1	1

5.3 BUILD UPB COMMAND

EXDOS will give this command for each unit immediately before it first attempts to access it. The command may also be given after a MEDIA CHECK command, depending on the response of the unit handler (see setion 5.4). The unit handler is required to build a complete UPB for the specified unit, and return a pointer to it to EXDOS. EXDOS will then copy the information which it requires from the UPB into its unit descriptor, and use it to control accessing of that unit until it asks for a new UPB. Since the data is copied it is not necessary for the unit handler to preserve the UPB after this one command, so it can use one data area for building UPBs for all units. EXDOS will check for the "VOL_ID" string and only use the volume id and dirty disk flag if it is present.

There is a special error code (.FXUPB) return allowed, to cope with the FORCE UPB command described in section 5.5. If the unit handler has been given a FORCE UPB command then any call to BUILD UPB should return .FXUPB to indicate the EXDOS that the same UPB is still valid. This can be returned in other circumstances if desired and EXDOS will continue to use its existing data.

PARAMETERS: B':IY = Segment and address of RAM area
A = 3
IX = Address of 512 byte scratch area
C = Relative unit number

RESULTS: A = Status code (normally 0)
B:DE = Pointer to byte zero of UPB

For a fixed media device, such as a hard disk or RAM-disk, or a device which only uses one format, the UPB can be fixed in the code making this command very simple. For a device such as UNITH which handles many different formats, it is necessary to determine from the disk itself what the format is. This generally involves reading the boot sector (first sector), which contains a copy of the UPB starting at offset 11.

For early disk formats (pre MS-DOS 2.00) this UPB is not present in the boot sector. The unit handler should therefore do some validity checks on the UPB and if these fail, it must determine the disk format from the FATID byte which is the first byte of the first sector of the FAT. In these formats this first sector of the FAT will always be the second sector of the disk (one reserved sector). The FATID byte must always be in the range F8h...FFh and corresponds directly with the standard media byte values listed in section 5.2.

The 512 byte scratch area in page-1 is provided for the unit handler to read the various sectors which it needs to in order to determine the UPB.

5.4 MEDIA CHECK COMMAND

The media check command is issued by EXDOS when it wants to check whether the disk in a particular unit (drive) has changed. The current media byte is passed to the unit handler and it must return one of three possible results:

0FFh => Media changed
0 => Not sure
1 => Media not changed

If "media not changed" is returned then EXDOS will take no further action, it will continue to use its current UPB and any buffered data. If "not sure" is returned then EXDOS will behave as if "media changed" was returned unless it has any dirty buffers for this unit in which case it will assume that the media has not changed. If "media changed" then EXDOS will call BUILD UPB to get the new disk parameters and volume id. If it determines that the disk has or may have changed and there are any dirty buffers then it will prompt the user with a warning message, otherwise it will carry on with the new disk.

A unit handler with non-removable media, such as a hard disk, can simply always return "media not changed". A removable media handler which can tell definitely when a disk has been changed (by a mechanism such as a latch on the drive door), can always return "media changed" or "media not changed" as appropriate. A handler which has removable media but no definite way of determining when a disk has been removed could get away with always returning "not sure" , but this would be very inefficient since it would continually have to re-read the boot sector to re-build the UPB.

A reasonable mechanism for devices like this is to implement a timer and remember the last time of access for each unit. When MEDIA CHECK is called, if this unit has not been accessed for at least two seconds or so then it should return "not sure", otherwise it should return "media not changed".

```
PARAMETERS:   B':IY = Segment and address of RAM area
                A = 2
               IX = Not used
                C = Relative unit number

   RESULTS:     A = Status code (normally 0)
                B = Media status - 0FFh => changed
                                   0  => not sure
                                   1  => not changed
```

A unit handler may discover that its disk has changed when it is called to read or write data. In this caes it can return the special error code .NOUPB which EXDOS will trap and will result in EXDOS going through its disk changed procedure as above (calling MEDIA CHECK and the BUILD UPB).

5.5 FORCE UPB COMMAND

This command is provided to allow the user to specify a disk format which the unit handler and EXDOS should use, to cope with disks which use a non standard but compatible format and do not have the UPB stored on disk. It is not necessary for a unit handler to support this command, if it does not then it should return a .ICMD error.

```
Parameters:     B =  0 to cancel forced UPB
                  =  1 to set up a foced UPB
                DE -> New UPB if B=1
Returns:        A =  Status code
```

The UPB should be checked for validity and a .NDOS error returned if it is invalid. Otherwise the unit handler should extract the information which it needs in order to access the disk, from the UPB and return with zero status code. Any call to BUILD UPB from now on must just return .FXUPB to tell EXDOS to continue using the same parameters.

The command will remain in effect until a FORCE UPB command with B=0 is received which should clear the FIXED_UPB flag and return the unit handler to normal automatic operation. After receiving this command the unit handler should return .NOUPB unit it has received a BUILD UPB command to log a new disk on.

6. READ, WRITE and WRITE-WITH-VERIFY COMMANDS

These three commands are used by EXDOS for all accessing of data from the disk, including file data, directories and FAT information. EXDOS regards the disk as being made up of a series of logical sectors numbered (0....N-1) where "N" is the "total number of sectors" from the UPB. Logical sector zero is always the boot sector. The usage of other sectors depends on the information in the UPB. All sectors are 512 bytes in size.

The READ, WRITE and WRITE-WITH-VERIFY commands are given a unit number, logical sector number and sector count. Data at the disk transfer address should be read from, written to, or verified against the disk starting at the specified logical sector and continuing for the specified number of sectors. EXDOS does not know about different tracks or sides of the disk so it is up to the unit handler to cope with the mapping of sectors onto the disk.

PARAMETERS: B':IY = Segment and address of RAM area
A = 3, 4 or 5
IX = Disk transfer address (page-1)
C = Relative unit number
B = sector count (1...255)
DE = Logical sector number (0....)

RESULTS: A = Status code (normally 0)
B = Number of sectors successfully transferred.

If an error occurs then the unit handler should make all appropriate retries before reporting the error back to EXDOS. If an error does occur then the value returned in register B will be less than the original sector count. It should be the number of sectors which were transferred before the error occurred. When an error is returned to EXDOS, it will be reported to the user, who will be given the option of retrying the operation. This retry will begin from the sector which failed, rather than repeating the whole operation.

7. ERROR CODES FROM UNIT HANDLERS

In principle a unit handler can return any EXOS error codes, or any others which it defines itself. However there are certain error codes which are defined by EXDOS which are particularly useful for unit handlers to return. These are:

.NOUPB - No unit parameter block. Returned when the unit handler wants EXDOS to call its BUILD UPB command either because no disk has been accessed or the disk has unexpectadly changed.

.FXUPB - Returned from BUILD UPB to force EXDOS to continue using its current UPB (normally returned while FORCE UPB is in operation).

.ICMD - Invalid unit handler command. If the unit handler gets invalid command code.

.IUNUM - Invalid unit number to unit handler. If the unit handler gets a unit number bigger than it thinks it is supporting.

.ISECT - Invalid sector number. If the sector number requested by EXDOS is larger than the total nuber of sectors on the disk. Also if the number of sectors requested would go beyond the end of the disk.

.NRDY - Not ready. Typically this means that there is no disk in the drive.

.VERFY - Verify error. On a write with verify command the data did not verify after a suitable number of retries.

.DATA - Data error. Typically this means a CRC or lost data error occurred in reading or writing a sector.

.RNF - Record not found. A particular sector could not be located on the disk. This can result from trying to access a double sided disk in a single sided drive.

.WPROT - Write protected disk. Any attempt to write to a disk with the write protect label on.

.UFORM - Unformatted disk. This should be returned if the unit handler has reason to believe that the disk it is accessing is unformatted.

.NDOS - Not a valid EXDOS disk. This should be returned by the BUILD UPB command if it cannot determine a valid UPB for the disk. Also returned by FORCE UPB if the UPB it was given is invalid.

++++++++++ END OF DOCUMENT ++++++++++

