Addendum to the SpartaDOS X 4.20 Reference Manual
addressing new features introduced in v.4.40

Chapter 2: Introduction to SpartaDOS X

Setting time and date

The default time and date format as of version 4.40 is the European format (dd-mm-yy). Optionally, users can select the US format (mm-dd-yy). Because of that, the **DATE** command now displays the message about the proper order of numbers to type:

```
Current date is 10-09-06
Enter new (DD-MM-YY):
```

Running programs

Preceding the program name with the # character has the same effect as the X command, for example, typing:

```
#DISKRX
```

at the DOS prompt will do the same as typing:

```
X DISKRX
```

Most programs written for AtariDOS should be executed using X.COM. Doing so is most of the time the proper way of getting rid of the troublesome „Memory conflict" error; if using X.COM does not cure that, you probably need to reconfigure the system (see Chapter 8 – Configuring Your System).

Hard disks users will probably be glad to hear, that it is now possible to configure the DOS so that it can, if necessary, disable the ROM module automatically while executing a program.

DOS 2 Equivalents

M – RUN AT ADDRESS	External command RUN

Device identifiers

As of version 4.40 the DOS can handle up to fifteen drives attached simultaneously to the computer. At the SIO level the drives are numbered from 1 to 15 ($01-$0F). Drives 1-9 have, just as in the earlier SpartaDOS X versions, identifiers 1: ... 9: or A: ... I: in the Command Processor, and D1: ... D9: or DA: ... DI: in BASIC.

The drives 10-15 have letter-identifiers only, i.e. J: ... O: in the Command Processor, and DJ: ... DO: in BASIC.

Apart from DSK:, CAR:, PRN: and COM: there is also the NUL: device in the system. It can accept any amount of data written to it (not saving it anywhere), and while read, it can behave in one of the three ways, as follows:

NUL: or NUL1: – returns error 136 (EOF).
NUL2: – an infinite number of zeros is returned.
NUL3: – an infinite number of random bytes is returned.

Disk Format Compatibility

The SpartaDOS X 4.40 disk format is a slightly modified version of the format known from the previous SpartaDOS releases. It is backward compatible with the format used in SpartaDOS 2.x/3.x/X 4.x, so there should be no problem exchanging data between all these SpartaDOS versions.

It is, however, not recommended to do any write operations (including file deletions) on SpartaDOS 1.1 diskettes, should they remain accessible to SpartaDOS 1.1 afterwards. The advice is to copy the files over to a SpartaDOS X disk first, and edit thereafter.

The SpartaDOS X 4.40 is able to build filesystems on disks formatted to 512 bytes per sector. This density, new to the Atari world, is called *DD 512*. Files saved on such a disk are not accessible to any other version of SpartaDOS (including any older version of SpartaDOS X), and, as far as we know, neither to any other DOS running on Atari.

APPEND

Purpose: to append the given path at the end of the $PATH variable.
Syntax: APPEND *pathname*
Type: external – on device CAR:

The command appends the given pathname at the end of the $PATH variable.

ARC

The bug, when ARC, if run with the XH option, was displaying a message to the user without enabling the ANTIC display, is now fixed. Also, it is now possible to answer (apart from „No" and „Yes") „All" to the question, whether the data being extracted should replace an already existing disk file with the same name.

BASIC

The BASIC command can now execute the internal Atari BASIC module even if the computer has 1088k RAM. This was not possible in previous SpartaDOS X versions.

The environment variable $BASIC has no default value now, unless RAMDISK.SYS is installed. If the $BASIC variable has not been set by the user (with the SET keyword in CONFIG.SYS) while RAMDISK.SYS is being installed, the ramdisk driver sets the variable so that it points to a BAS.SAV file residing in the ramdisk.

BLOAD

Purpose: to load the given file into the given memory area starting at the given address.
Syntax: BLOAD fname.ext [$]address
Type: external – on device CAR:

The file *fname.ext* is loaded as a raw data block into the specified area, and then the control is handed back to the Command Processor.

There are no checks done, whether the file fits in the memory, or if vital operating system areas are safe – it is assumed, that the user invokes the command on purpose.

See also LOAD.

CHKDSK

In the previous versions of the SpartaDOS X the CHKDSK was a command internal to the Command Processor. Now typing CHKDSK at the DOS prompt invokes the program called CHKDSK.COM residing on the CAR: device (CAR:CHKDSK.COM).

Adding **/X** to the command causes an extended disk information to be displayed. Additionally, on **/V**, the disk bitmap (VTOC) is loaded into the memory and analysed, and the information about the remaining free space is compared with the amount of free space indicated by the bootsector. This allows to check quickly, if there are lost sectors on the disk. The /V will only work with the regular SpartaDOS disks.

COLD

In the Maxflash versions of the SpartaDOS X v.4.40 the **COLD /C** command is an equivalent to the **COLD** alone (without the parameter).

COMP

Purpose: to compare the given files.
Syntax: COMP [d:][path]fname1.ext [d:][path]fname2.ext
Type: external – on device CAR:

The program compares both files and displays information about the differences.

COPY

COPY is a command internal to the Command Processor. The only thing it does, however, is to launch CAR:COPY.COM. Now it is possible to replace this one with an arbitrary program. The environment variable $COPY is used for that. For example:

```
SET COPY=C:>SYS>CP.COM
```

causes the Command Processor to launch the indicated program instead of its defaults, and to pass all of the user-specified parameters to it.

DATE

The DOS now displays time and date in the European (dd-mm-yy) or American format (mm-dd-yy) depending on user selection. This applies to the DATE command as well. To change the format you should set the $DAYTIME variable in the environment. Its meaning is as follows:

```
SET DAYTIME=1
```

selects the American format, and

```
SET DAYTIME=2
```

the European one. Lack of the variable or DAYTIME=0 selects the default format – i.e. the European one in the current version of the DOS.

DELTREE

Purpose: to delete subdirectory trees recursively.
Syntax: DELTREE [d:][path]dirname
Type: external – on device CAR:

The command when executed asks for the confirmation, and, if confirmed, removes the given subdirectory recursively with all the content, reporting progress successively.

If it aborts displaying *Can't delete directory*, that means, that there is an invalid directory entry on the disk – a file, that was opened for writing, but never closed for a reason (e.g. because of a system crash). Such an entry is invisible in directory listings and cannot be normally deleted, but even when all the files inside a directory have been removed, due to the presence of such an invalid entry, the directory is not considered empty and cannot be deleted. This is an error condition that indicates that the filesystem structure is not completely valid: you should run the CleanUp program to verify the filesystem structure and fix it.

DF

Purpose: to display summary information about free space on all disks.
Syntax: DF
Type: external – on device CAR:

The command produces a list of all active drives, displaying the following information for every single drive: the drive letter, the total number of sectors, the number of free sectors, the number of free kilobytes, the percent of the disk used, the volume name. A summary is displayed at the end.

DIR and DIRS

Earlier SpartaDOS versions display six last digits of the file size information in directory listings, even though the file size can be an 8-

digit number, and such long files can be created and are correctly handled (despite this flaw). Thus, it is rather difficult to properly estimate the size of some long files.

SpartaDOS X 4.40 solves this problem: when a file exceeds 999999 bytes in size, this number is displayed in kilobytes, and to indicate this, there is a ‘k’ character printed at the end. Example:

```
Volume:      TEST
Directory:  MAIN

BACKUP   TAR   6988k 10-09-06 15:55
  51483 FREE SECTORS
```

Short directory listing obtained by DIRS contains subdirectory extensions (instead of DIR). A colon is used to indicate a subdirectory, like in MyDOS.

The switch /A displays file attributes in the list. It is used mostly together with “+”, for example `DIR +/A` command shows all files with their attributes.

DPOKE

The DPOKE is similar to POKE, except that it puts a two-byte value (a word) into the given address, preserving the usual 6502 byte order (low/high).

DUMP

The /A switch is added, that causes some ATASCII-specific characters (semigraphics, inverse video characters etc.) to be replaced with dots. This allows to print the DUMP output on a printer even, if the printer interface does not allow full translation or if it's not using a graphics mode to print.

ECHO

Purpose: to enable or disable the „echo” in the Command Processor.
Syntax: ECHO ON|OFF
Type: internal, executed by COMMAND.COM.

ECHO OFF disables echoing user commands, passed to the Command Processor from the command line or fetched from a batch file. ECHO ON restores the default.

ED

Purpose: a text editor.
Syntax: ED [d:][path][filename.ext]
Type: external – on device CAR:

ED.COM is a SpartaDOS X-compliant, relocatable version of JBW Edit. The main purpose of the program is to edit the DOS configuration files, but it obviously can be used to edit any text files, if they are not too big (the practical file size limit is about 6-8 kB, even if the editor buffer is much larger).

When there is a filename given on the command line, the program will attempt to load it. There are the following editing commands available:

Esc – cancel the function or quit the program.
Ctrl/L – Load – load a file into the buffer.
Ctrl/S – Save – save the buffer to a file.
Ctrl/U – Up – move up the low margin of the editor window.
Ctrl/D – Down – move down the low margin of the editor window.
Ctrl/V – Visible – make EOL characters visible.
Ctrl/B – Begin – move the cursor to the beginning of the text.
Ctrl/E – End – move the cursor to the end of the text.
Ctrl/A – move the cursor to the beginning of the line.
Ctrl/Z – move the cursor to the end of the line.
Ctrl/T – Tag – tag the current line.
Ctrl/G – Go – move the cursor to the tagged line.
Ctrl/Q – Quit – quit the control mode, the next key combination will be interpreted as a character.
Shift/Ctrl/up arrow – page up.
Shift/Ctrl/down arrow – page down.
Shift/Insert – insert the current line before the tagged one (see Ctrl/T), the cursor moves to the next line.
Shift/Ctrl/E – Erase – clear the editing buffer.

ELSE

See Chapter 5, section „Batch files”.

EXIT

See Chapter 5, section „Batch files”.

FI

See Chapter 5, section „Batch files”.

FORMAT

The formatter has been enhanced to support larger filesystems (up to 32 MB per disk is now possible), to handle greater sectors (up to 512 bytes), and to be able to use more drives. The drives are selected by the U function (as in *Unit*) using letters from A (drive 1) to O (drive 15). The density selection offers, apart from the three old ones, i.e. *Single, Dual, Double*, the fourth density named *DD 512*. This is the double density at 512 bytes per sector (just as the *Dual* is in fact double with 128 bytes per sector). The *DD 512* density is usable with certain types of hard drives (KMK/JŻ/IDEa), and with TOMS floppy drives, either the original ones (TOMS 710, TOMS 720), or third party drives with TOMS Turbo Drive or TOMS Multi Drive extensions installed.

There is also a new option there – *Optimize*. The earlier SpartaDOS versions build directories in such a way that the last sector on the disk is marked as occupied and left unused. When the *Optimize* option is enabled in the formatter, that sector is reclaimed and assigned to the data area, so that you have one free sector more, than you usually have on a freshly formatted diskette.

There are some ancient hard drives however, which are physically formatted so that the very first sector of a partition has number 0, and not 1, as it should be on an Atari disk. This 0 sector is not accessible, but it does count into the summary of existing sectors returned to the computer by the disk controller. This problem occurs e.g. in Supra Corp. and K-Products drives. On such a drive, the sector 'reclaimed' by the formatter's 'Optimize' function does not really exist. If you have such a drive, you should keep the 'Optimize' off when building directories.

Caution: the FORMAT command uses the 6502 stack space intensively. Because of that, some floppy turbo systems, which load the fast serial I/O patch onto the stack, will not work in turbo mode. Problems will occur with Top Drive 1050, TOMS Turbo Drive or any other using the same method. Such a drive can be used with SpartaDOS X, but only at the „slow" (i.e. standard) baudrate.

Other turbo systems, such as TOMS Multi Drive (in the Ultra Speed mode), TOMS 710/720, CA-2001, Atari XF551, LDW 2000 Super, Indus GT, Happy 1050, US Doubler – will work normally.

FSTRUCT

Purpose: to analyse a binary file
Syntax: FSTRUCT [d:][path]filename.ext
Type: external – on device CAR:

The command displays the information about the structure of the specified binary file (type, load address, length of the segments etc.).

FSYMBOL

Purpose: to convert a symbol to an address.
Syntax: FSYMBOL symbol_name
Type: external – on device CAR:

The command displays the address associated with the specified symbol. See also SL.

IF

See Chapter 5, section „Batch files".

GOSUB and GOTO

See Chapter 5, section „Batch files".

MAN

Purpose: documentation viewer
Syntax: MAN [filename]
Type: external – on device CAR:

MAN.COM is a simple text viewer. Its main purpose is to view documentation files to programs and commands given as the parameter. The program's operation is similar to the UNIX' *man* command (where it is an abbreviation for *manual*), thus the name.

To use it, it is required to define $MANPATH first in the environment. The variable will contain a list of directories to be searched for text files, in an identical manner as $PATH contains a list of directories to be searched for executables:

```
SET MANPATH=C:>MAN;D:>DOC
```

Now if you put to any of these directories a file with *.MAN, *.DOC or *.TXT extension, and then give its name (but omitting the extension) to MAN.COM, the file will get searched for and displayed when found.

For example, the HDSC.ARC archive contains a text file named HDSC.DOC filled with informations about the program. When you have unpacked the archive, you can put the executable to any directory pointed to by $PATH, and the *.DOC file to any of the directories pointed to by $MANPATH. Now, when you want to read the instructions, you do not need to remember, where the original archive is, unpack it again etc. It is enough to execute this command instead:

```
MAN HDSC
```

and the HDSC.DOC appears on the screen.

If the text file has „long lines", or in an extreme case the entire file consists of a single line, the viewer will try to justify the text so that it fits to the current width of the screen.

MAP

Purpose: SIO.SYS control.
Syntax: MAP [unit] [SIO|OS|NORMAL|OFF] [d:]
or: MAP [filename.ext]
Type: external – on device CAR:

The command „maps" the given drive identifier (*d:*) to the drive associated with the specified number (*unit*). An internal SIO translation table is used here. The additional options control the communication mode for the specified disk:

NORMAL – standard mode, the PBI has priority over the SIO.
SIO – SIO communications only (the PBI is bypassed).
OS – the communication is redirected to the ROM OS.
OFF – drive disabled (or handled by another driver).

The „SIO" option allows to gain access to a serial floppy drive, which has been masked out by a parallel (PBI) drive having the same number. For instance, the command `MAP 1 SIO D2:` creates a logical D2: drive, which uses physical disk drive number 1 (the D1: can be a partition of a parallel hard drive).

The „OS" parameter applies, when the OS ROM routines are to be used instead of the native SpartaDOS communication routines. „OS" cannot be used, when the DOS is configured to USE OSRAM mode.

The SIO parameters to be changed can be stored in a file, and the name of that file can be given to the MAP command as the first (and only) argument. That should be a text file, and each line should contain correct MAP parameters starting from the *unit* value. This feature allows for changing the settings of multiple drives at once.

Caution: MAP does not work, when SIOOLD.SYS is installed instead of SIO.SYS!

MDUMP

Purpose: to display memory in hex and ATASCII.
Syntax: MDUMP [$]address
Type: external – on device CAR:

The command does the same to memory, what DUMP does to files. It is useful to check the memory contents quickly.

MEM

Purpose: to display memory information.
Syntax: MEM [/X]
Type: external – on device CAR:

The MEM command is now external, using the program that resides on the CAR: device (CAR:MEM.COM). MEM now displays additional information on the current memory configuration, i.e. the mode being used by the DOS (NONE, OSRAM or BANKED). The additional /X switch makes the information a bit more detailed.

The information about the number of free banks may differ from what was displayed by the previous versions of SpartaDOS X. The reason is that the SpartaDOS X 4.2x was not able to properly recognize some types of RAM expansions, and because of that, for example 256k TOMS (12 banks) was seen as 192k (8 banks). As of 4.40 the memory handler is fixed.

PROC

See Chapter 5, section „Batch files”.

RENAME

The RENAME command and the corresponding kernel function have been improved so that you should not be able to give a name to a file, when such a name already exists in the same directory.

RENDIR

Purpose: to rename directories.
Syntax: RENDIR [d:][path]old_name new_name
Type: external – on device CAR:

The command does the same to directories, what RENAME does to files.

RETURN

See Chapter 5, section „Batch files”.

RUN

Purpose: to run the code at the specified address.
Syntax: RUN [$]address
Type: external – on device CAR:

The command makes a JMP to the specified address. It is not sanity-checked before, it is assumed, that the user is invoking the command on purpose.

SETERRNO

See Chapter 5, section „Batch files".

SIOSET

Purpose: SIO.SYS serial speed control.
Syntax: SIOSET [d: [type [usindex]]]
Type: external – on device CAR:

The SIOSET purpose is advanced serial protocol control for the SIO.SYS driver. Typically the serial transmission parameters are determined automatically and there is no need to change them. Sometimes however (e.g. when a drive was changed to another type at runtime) you may want to change them by hand.

With no arguments given, SIOSET displays the current configuration for all drives. The *type* parameter has the following meaning:

RESET – the transmission parameters for the drive are cleared; they will be determined on next I/O request sent to that drive.
NORMAL – the drive works at standard baudrate.
XF – the drive uses the XF551 protocol.
US – the drive uses the UltraSpeed protocol.
INDUS – the drive uses the Indus protocol.

When UltraSpeed is used, the additional *usindex* parameter allows to determine the serial speed. For example, the so called 3xSIO mode (3x19200, i.e. 57,6 kbps) requires $08 as *usindex* value.

Caution: SIOSET does not work, if SIOOLD.SYS was loaded instead of SIO.SYS!

SL

Purpose: generates a list of SpartaDOS X symbols.
Syntax: SL

Type: external – on device CAR:

The command generates the list of SpartaDOS X symbols.

SORTDIR

Purpose: to sort filenames in directories by name, type, date or size.
Syntax: SORTDIR [d:][path] [/NTSDX]
Type: external – on device CAR:

The command reads the specified directory, sorts it using the specified criteria, and then writes it back. The criteria can be:

/N – sort by name
/T – sort by type
/S – sort by size
/D – sort by date and time
/X – in descending order

The file specification may be omitted, the current directory is sorted then, but a criterion is obligatory. SORTDIR invoked without arguments displays a brief copyright information and lists available options.

When the files are sorted by name, the file type is a second priority. When sorting by type, the second priority is the file name. When sorting by size, the second priority is the name, and the type is the third. Digits are prior to letters. Everything is sorted in ascending order by default, the /X switch reverses that order.

SWAP

The SWAP command has no effect on drives from J: to O:.

TD

The TD command has been fixed (the „Y2K problem" is fixed). The format in which the time is displayed depends on the value of the DAYTIME variable (see DATE command). Additionally, the „X" letter reflects the Caps/Inverse state. See also Z.SYS (chapter 8).

UNERASE

UNERASE.COM in SpartaDOS X 4.20 contains a long-known bug, that causes it to screw up the disk's bitmap, when the file being undeleted resides in a series of data sectors, and the group of bits representing these sectors happens to cross a sector boundary in the bitmap. This bug is

fixed, the test distributed with Nelson Nieves' NNTOOLS now passes without errors.

VDEL

Purpose: delete selectively in groups of files.
Syntax: VDEL [d:][path]filename.ext
Type: external – on device CAR:

The command finds a file matching the specified filename mask, displays its name and asks, if the file should be deleted. The procedure is repeated until no file matches the mask anymore. At the end, a brief summary is displayed. The program can be aborted with the **Esc** key.

VDEL is useful in the (rare) cases, when in a group of many files, there are some to be deleted, and none of them share a common filename mask with the others. The files must be then deleted one by one, and VDEL, at least partially, facilitates this task. It can be yet more conveniently done with the MENU program.

XFCONF

Purpose: density selection in floppy drives.
Syntax: XFCONF [d:] [/12345]
Type: external – on device CAR:

Some floppy disk drives have problems when it comes to automatic density detection. This problem particularly beats the Atari XF551 disk drive (thus the name of the command), but the program can be handy in case of any other drive too, when it happens to be somehow stuck in an improper density. Then, the XFCONF allows to force the drive into the proper one by hand.

The command invoked without arguments displays menus, where you can choose drive number (1-4) and the desired density. After the new configuration was sent out to the drive, it is checked, whether the drive has really selected it (some drives do accept and positively acknowledge densities impossible for them, for example 360k for an Atari 1050 Top Drive, or 720k for an Atari XF551 – the drive's controller selects something closest to the requested configuration and replies „OK" to the computer – which is surely far from being OK). The message *Drive cannot do this density* means, that the requested density was accepted, but the drive is in fact unable to realise it.

Other messages:

Drive not configurable – the drive is a stock Atari 810 or Atari 1050, without modifications. It is not possible to change densities.

Drive is not a floppy disk – an attempt to change density of a ramdisk

or harddisk.

Drive rejected the density – the drive explicitly denied to select the density.

XFCONF accepts command line arguments too. Giving a drive identifier alone causes the computer to check, if the drive is a floppy drive and if it is configurable. The reply *Drive is configurable* confirms that. The density change can be accomplished by adding a switch followed by a digit from 1 to 5, where the digits mean densities as follows:

/1 – single density (SSSD, 90k)
/2 – dual density (SSED, 130k)
/3 – double density (SSDD, 180k)
/4 – double sided double density, 40 tracks (DSDD, 360k)
/5 – double sided double density, 80 tracks (DSDD, 720k)

Other commands

Apart from the commands mentioned above, there are two programs on the CAR: device, written for hard disk interface users:

S2I – for SIO2IDE
MNT – for KMK/JŻ IDE and IDEa

The description of these two programs belongs to the documentation of the respective hardware and thus will not be discussed here.

Running programs

If a program requires free memory in the area normally occupied by the I/O library ($A000-$BFFF), it should be executed with the X command. You can also precede the program name with the hash mark instead, for example, typing at the DOS prompt:

```
#DISKRX
```

will have the same effect as:

```
X DISKRX
```

BATCH FILES

A text line starting with a semicolon (;) is understood as a comment and skipped over without parsing.

The command ECHO OFF used in a batch file prevents displaying the commands read from the batch file. ECHO ON enables the echoing.

As of SpartaDOS version 4.41 batch files are executed with ECHO OFF by default: you have to write ECHO ON explicitly in the batch file, if you want to see what is being done.

Conditionals

The external commands IF, ELSE, FI allow simple conditional expressions in batch files. The general syntax is:

IF [NOT] EXISTS [atr] pathname|ERROR [n]|INKEY [‘c’|n]
...
FI

or

IF [NOT] EXISTS [atr] pathname|ERROR [n]|INKEY [‘c’|n]
...
ELSE
...
FI

Such conditionals can be nested, up to 255 levels of nesting is allowed.

The IF EXISTS conditional

The operator EXISTS allows to check the presence of the specified file or directory on a disk. For example, IF EXISTS FOO.BAR is true, if a file named „FOO.BAR" exists in the current directory. NOT negates the result, so IF NOT EXISTS FOO.BAR will be false in this case. It can only see regular files by default, to check if a directory exists, you have to specify the attribute the usual way: IF EXISTS +S FOO returns true, if a subdirectory named „FOO" exists in the current directory. Example usage:

```
IF EXISTS %1.ARC
 IF NOT EXISTS +S %1
  ECHO Creating dir %1
  MD %1
  ARC X %1 %1>
 ELSE
  ECHO %1 already exists
 FI
ELSE
 EXIT 170
FI
```

Save this as a batch file named for example X.BAT somewhere in your $PATH. Then, typing at the command prompt:

```
-X FILES
```

allows to unpack the archive FILES.ARC into a subdirectory automatically created on the purpose.

The IF ERROR conditional

The operator ERROR allows to check error conditions. IF ERROR is true, if any error has been recorded by the system. Similarly, IF NOT ERROR is true, when there was no error condition. Alternatively you can specify the exact error number: IF ERROR 170 returns true, if the error that occurred last was „File not found".

The ERROR keyword uses a system variable called ERRNO, that is only written to by the system library on any error condition, and never cleared. This means, that the error code the variable contains may not necessarily have been generated by the command that was executed last. It is therefore a good practice to do SETERRNO 0 before running a command that is about to be checked later with IF ERROR.

The IF INKEY conditional

When this operator is encountered, the batch file execution stops and the system waits for a key to be pressed. The expression IF INKEY is true, when the user hits any key (except Break – it is false then). You may

also specify the key to be waited for. To accomplish that, an argument must be specified to the INKEY keyword, either a numeric ATASCII code of the key, or its text value in single quotes. For example, when it is the „A”-key you want the batch file to wait for, the command doing that may be in any of the form below:

```
IF INKEY 65
IF INKEY $41
IF INKEY 'A'
```

This serves well in simple comparisons, but for more complex tasks, like for example a menu with many options, you should use the INKEY command described below.

Comparisons

Some parts of a batch file can be executed or not depending on the value of an environment variable. Checking the value is accomplished with the equation sign (=), which is employed here as a comparator. For example, IF DAYTIME=2 is true, if the environment variable DAYTIME contains the text „2”. There is no separate „not equal” operator, this condition can be checked by combining the equation sign and the logical negator NOT. So, IF NOT DAYTIME=2 means „if DAYTIME is not equal to 2”.

GOTO jumps

The GOTO command allows to make a jump within the batch file. The syntax is:

```
GOTO label
```

This simply transfers the batch file execution to the line following the one, that contains the „label” at its beginning. An example definition of a label may look as follows:

```
:LABEL
```

That is the definition. Giving the label's name as a parameter to the GOTO keyword you have to omit the colon.

Bear in mind, that a GOTO searching for its label always goes through the entire batch file from its beginning. This means, that the farer within the batch file is the label, the slower a GOTO jump will be.

The INKEY command

This command stops the BAT file execution, waits for a key to be pressed, and, when pressed, assigns the corresponding letter to the specified environment variable. This value can be read within a comparison in the IF statement, and so you can this way make for example a menu with more options:

```
:MENU
CLS
ECHO A. Option no. 1
ECHO B. Option no. 2
ECHO C. End
INKEY KEY
IF KEY=A
   ECHO Option 1 was selected
FI
IF KEY=B
   ECHO Option 2 was selected
FI
IF KEY=C
   SET KEY
   EXIT
FI
PAUSE
GOTO MENU
```

Caution: data is written to the environment, and the environment space is only 256 bytes. To avoid filling it with unnecessary garbage, it is a good practice to delete all variables created in your batch file before exiting it; just as it is shown in the example above (the fifth line counting from the end).

The INKEY command should not be confused with the INKEY operator, which is a part of the IF command.

Procedures

It is now possible to define a procedure within a batch file. Functionally, a procedure corresponds to a subroutine in Atari BASIC or a procedure in Turbo-BASIC XL. The definition of a procedure begins with PROC, and ends with RETURN, in the following manner:

PROC name
...
RETURN

Such a procedure can be called with:

GOSUB name

Alternatively, GOSUB can also call ordinary GOTO-labels (see above), provided the command sequence marked so is ended with

RETURN.

Procedure calls can be nested, the maximum number of nested calls ever possible is 20. This number however can be limited down by other factors, so we do not recommend to exceed some 8 levels of nesting.

Other commands

The EXIT command causes an immediate termination of the batch file processing. Alternatively you can add an exit code to be taken by the system as an error code. For example, EXIT 170 causes the batch file to be terminated with „File not found" error.

The SETERRNO command causes the ERRNO system variable to be overwritten with the given value. For example:

SETERRNO 170

will set 170 as the last-occurred error in the system. The keyword's most obvious usage is to clear the variable before execution of a command, that is about to be controlled later with the IF ERROR conditional.

Other comments

While a batch file is being executed, the Command Processor is periodically loaded to the memory and unloaded (it is not a resident program). This has some negative influence on interpretation speed. To speedup this process, the Command Processor can be held in the memory whilst the batch file is being interpreted. You can accomplish that by adding the command LOAD COMMAND.COM at the beginning of your batch file. Before the batch file exits, the Command Processor can be unloaded with LOAD alone.

Set file position – POINT

The internal SpartaDOS X routine, corresponding to this function, has been rewritten, so that random access to very long files (greater than 500k) should now work up to four times faster than before.

Load binary file (LOAD)

Syntax: XIO 40,#IOCB,4,X,"D:[path]fname.ext"

If X is 0, the file will be loaded to the memory and executed. If X is 128, the file will be loaded without execution.

Get disk information (CHKDSK)

This function is slightly changed, due to the increased number of supported densities. The earlier versions of SpartaDOS support disks with 128- and 256-byte sectors – SpartaDOS X v. 4.40 adds support for 512-byte sectors, and theoretically even larger.

The CHKDSK function returns 17 bytes described in the SpartaDOS X Reference Manual. The byte holding the information about the sector size, found at the offset *buffer+1*, has a value of 128 for the 128 BPS densities (*Single*, *Dual*) and 0 for 256 BPS (*Double*). Originally it is, of course, just the low byte of the actual sector size value (128 = $0080, 256 = $0100) – and encoding sector sizes larger than 256 bytes is impossible this way.

Because of that, the interpretation of this byte in the SpartaDOS X v. 4.40 has changed, both inside the DOS itself and in its external utilities. The new way is of course backward compatible, the „old" values still retain their traditional meaning.

A value of 128 indicates, just as before, 128 BPS. Any other value is the high byte of the logical sector size value in bytes, *minus 1*. This allows to easily encode sector sizes from 128 bytes to 64 kilobytes, as follows ('*' – not supported):

Size (B)	Hex value	Encoded as	Remarks
128	$0080	$80 (128)	For backward-compatibility.
256	$0100	$00 (0)	As in 2.0 format.
512	$0200	$01 (1)	
*1024	$0400	$03 (3)	
*2048	$0800	$07 (7)	
*4096	$1000	$0F (15)	

*8192	$2000	$1F (31)
*16384	$4000	$3F (63)
*32768	$8000	$7F (127)
*65536	$0000	$FF (255)

An assembly routine that calculates the real sector size value out of the „encoded" one can for example look like this:

```
; the input code is given in the
; accumulator (A), the result
; is returned in A (low byte)
; and X (high byte)
;
getssize
        ldx #$00
        cmp #$80
        beq quit
        tax
        inx
        lda #$00
quit    rts
```

SpartaDOS User Accessible Data Table (COMTAB)

DECOUT2 COMTAB-21

Contains the right-justified, space-padded output of the *misc_conv32* routine, an ASCII string representation of the four byte number at DIVEND (see Page Seven „Kernel" Values). 10 bytes (including 8 byte DECOUT).

DIVEND COMTAB-6

A four byte number here will be converted by the *misc_conv32* routine to a string at DECOUT2. See Page Seven „Kernel" Values.

ODATER	COMTAB+19	3 bytes
OTIMER	COMTAB+22	3 bytes
TDOVER	COMTAB+25	1 byte

The function of these registers is similar to the function of DATE, TIME and DATESET registers respectively, except the TDOVER not being automatically cleared after use (unlike DATESET).

ODATER, OTIMER and TDOVER are the old date/time stamp control registers used on SpartaDOS 3.x. SpartaDOS X disabled them and replaced with the new triad of DATE/TIME/DATESET. This was the reason, why SpartaDOS 3.2 copy programs, when ran under SpartaDOS X 4.2x, were not able to control timestamps in the files copied.

SpartaDOS X 4.40 restores the function of these registers, but DATESET still has the highest priority, and when it is set, the TDOVER value is ignored.

The TDOVER is cleared after the program quits in SpartaDOS X 4.40,

but not in 3.2. Thus it is a good programming practice to clear this register always when the program is about to quit to DOS (of course, only if TDOVER was used).

Decoding the drive identifier

As it was already said in the preceding sections of this addendum, one of the new facilities offered by SpartaDOS X 4.40 is an increased number of drive identifiers. In other words, when other DOS-es support eight or nine disks (numbered from D1: to D9:), SpartaDOS X 4.40 supports fifteen. The additional drives (above D9:) are identified by drive letters from DJ: to DO:, following the convenience known from the previous SpartaDOS X versions, namely that the drive 1 can be referenced either as D1: or DA:, drive 2 as D2: or DB: and so on.

When a program receives a drive identifier from the DOS (e.g. as a command line input) and passes it on to another DOS function unchanged, there should be no problems with the new, „non-standard" drive identifiers. There will be no difference in the code either, when a program wants to calculate the real (binary) drive number – for DUNIT for example. To do that, it is enough to clear the high nibble of the drive digit or letter with an AND #$0F.

A reverse conversion may be a problem, though, because a $30 should be added to a DUNIT value to get a drive digit, or a $40 to get a drive letter. It was not necessary to distinguish these two cases so far, so the programs doing such a conversion on purpose will probably not work correctly on „new" drives (10-15). Luckily such a calculation is rarely necessary, but we supply an example just in case:

```
dsk2asc
        lda dunit
        ora #$30
        cmp #'9+1
        bcc ok
        adc #$0f
ok      rts
```

The result – an ASCII character symbolizing the given drive – is returned in the accumulator. It will work with any DOS. But if the program is to be used with SpartaDOS X only, the routine may be greatly simplified:

```
dsk2asc
        lda dunit
        ora #$40
        rts
```

Symbols

A symbol is an eight-character name of an object residing somewhere

in the computer's memory. Such an object can be a routine or a data structure. When the symbol name is known to the programmer, it can be translated inside the program to the actual address. Normal binary programs have to do that „by hand", i.e. making the appropriate system call (see Page 7 „Kernel" Values). This is neither the most convenient nor the only way to do that; some assemblers can generate SpartaDOS-specific, specially structured binaries; in case of such a binary the symbol-to-address translation is done automatically by the loader.

If a symbol exists, that means that the corresponding routine is loaded into the memory, and the symbol itself provides the information, where it can be found. Addresses pointed to by symbols can change depending on SpartaDOS version or the order of loading drivers. A part of the symbols points to the ROM, part of them is even defined in ROM, but even in such a case they cannot be considered fixed forever – every symbol can, at any moment, get replaced by its new instance pointing to another place in the memory. This happens when a device driver replaces or patches a system procedure.

If a symbol does not exist, most of the time it means that the corresponding driver is not loaded to the memory.

Vectors Under the OS ROM

The vectors are created under the OS ROM to secure some backward compatibility with SpartaDOS 3.2 and earlier versions. There is no need to use them in SpartaDOS X, as the same routines are pointed to by appropriate symbols, so you do not need to look under the OS ROM or worry, if the vectors still exist, or were destroyed by a program loaded in the meanwhile (Turbo BASIC XL, for instance).

Here is a list of symbols corresponding to the old vectors:

Vector	Label	Symbol	Defined by
$FFC0	VGETTD	I_GETTD	clock drivers (e.g. CLOCK.SYS)
$FFC3	VSETTD	I_SETTD	as above
$FFC6	VTDON	I_TDON	TD.COM
$FFC9	VFMTTD	I_FMTTD	TD.COM
$FFCC†	VINITZ	_INITZ	kernel
$FFCF†	VINITZ2	-	-
$FFD2	VXCOMLI	-	-
$FFD5†	VCOMND	-	-
$FFD8†	VPRINT	PRINTF	kernel
$FFDB†	VKEYON	I_KEYON	KEY.COM

The vectors are not used by SpartaDOS X itself, they can also be accidentally destroyed by software using the RAM under the OS ROM,

so their use implies some trouble. In future versions of SpartaDOS X they may disappear completely (the cross in the table marks the locations already obsolete in SpartaDOS X 4.20).

The symbols I_GETTD, I_SETTD, I_TDON and I_KEYON work the same way as the old vectors VGETTD, VSETTD, VTDON and VKEYON described in the SpartaDOS X Reference Manual. There is only one exception, namely that the lack of the appropriate driver being loaded is known not from the Carry state after the call, but rather from the impossibility to do the call due to inability to find the symbol.

In the I_GETTD and I_SETTD procedures, the Carry, when set, means that the clock driver is busy at the moment, you should call the routine again. It is a good programming practice to count, how many times in a row the call failed, and break the loop after a certain number (e.g. 255) of iterations to avoid deadlock, when the clock becomes unresponsive for a reason (e.g. a hardware failure).

The I_FMTTD is loaded to the memory along with the TD.COM, being a part of it. It accepts an address of a 32-character buffer in the Y/X (low/high) registers. When the call returns with Carry set, this means an error (the clock driver being in permanent busy state). If the Carry is cleared, there is time and date information in the buffer, in a form of ASCII, EOL-terminated string.

Page 7 „Kernel" Values

1) The *misc* vector

The original SpartaDOS X Reference Manual mistakenly prints „4" as a function code for the *misc_convdc*, when it is in fact 5. This addendum gives an occasion to correct that error.

A fact of some importance is that the conversion routine behind this function code is now 32-bit, with the most significant byte of the DIVEND at COMTAB-3, and generates resulting 10-character ASCII strings at DECOUT2. But to retain the compatibility with existing applications the old *misc_convdc* entry zeroes that highest byte before proceeding with the conversion, thus cutting the number down to 24 bits and the result to 8 digits. For 32-bit conversions a new entry should be used, labelled *misc_conv32*, with function code 11. This function code is available only as of SpartaDOS X 4.40, calling it on an earlier version of the DOS will cause undefined results.

2) The *device* register

The device code $6x is assigned to the NUL: device.

3) The *block_io* vector

block_io $0706

The *block_io* routines support reading and writing sectors. Using this vector instead of *lsio* decreases the number of DCB variables to set in the program and greatly reduces worries about disk density and the combinations of the sector number and its size (in the double density, as it is widely known, the first three sectors are 128-bytes wide each, while the other sectors in this density are 256-byte each).

Before a call to *block_io* you should set the following DCB variables: the device code (DDEVIC), the device number (DUNIT), the buffer address (DBUFA) and the sector number (DAUX1/2). The function code should be passed in Y. Upon exit, the system puts the status code into the accumulator: 0 or 1 for a success, or a negative error code otherwise.

The *block_io* function codes are here:

0 – *bio_rdsec* – read sector (standard, no density check)
1 – *bio_wrsec* – write sector (as above)
4 – *bio_rdsys* – check density, remember it, and read sector
5 – *bio_sbps* – remember the sector size currently set in DBYT

Other function codes (2 and 3) are reserved for internal use of the SPARTA.SYS driver.

Functions number 0 and 4 operate similarly, except that the latter fetches information about the actual density from the drive first, and stores it into a memory table for later reference. The functions numbered 0 and 1 use that information, and so a program, that wants to access sectors this way, must always call the function number 4 first, e.g. to read sector number 1, and use the function 0 to read all other sectors.

If it is necessary to bypass the density recognition mechanism, and the sector size is otherwise known, in lieu of the function 4 one can store the required sector size to the DBYT variable ($0308-$0309), the required drive number to DUNIT ($0301), call function 5 and then subsequently 0.

4) The *fsymbol* routine

fsymbol $07EB
ext_on $07F1
ext_off $07F4

One of the new things in SpartaDOS X 4.40 is the *fsymbol* routine, which can translate symbols into memory information (an address and memory code). Finding symbols with this routine is very simple: the address of the symbol name should be passed in AX (low/high). This name must always be eight characters long, if it is shorter, pad it with

spaces.

Upon return, when the Z flag is set, then there is no symbol defined of that name. Otherwise the AX registers contain the address associated with (pointed to by) the symbol, and the Y register – the memory code. This code should be given (in accumulator) to the *ext_on* routine, which configures the memory appropriately (if necessary) to make the address available. Afterwards the *ext_off* should be called to restore the memory to the previous state to the *ext_on* call.

5) The *sio_index* table

sio_index $070F

The *sio_index* table, changed by the CP's SWAP command and referenced by SIO drivers, is still 9 bytes wide and cannot be enlarged, even though the number of valid SIO drive numbers has been increased to 15. This is the reason why the drives 10-15 cannot be „swapped".

Chapter 7: Technical information

Boot sectors

In the standard Atari densities, i.e. 128 and 256 BPS, the first three sectors of a disk have 128 bytes each and are occupied by the SpartaDOS bootloader. This bootloader is not significantly changed in SpartaDOS X v. 4.40 and is almost identical to the one found in earlier revisions.

The new thing is, that when data sectors on the disk are greater than 256 bytes, the first three sectors are of the same size as the others. Moreover, the boot region takes only one sector, the one number 1. The first 42 bytes of this sector carry information about the filesystem structure, just like in earlier versions of the SpartaDOS. The remaining portion of the sector is occupied by the new bootloader, able to handle 512-byte sectors and greater ones.

It can be thought, that such a disk structure (and de facto breaking the existing standard) is an unnecessary complication. Indeed, in SpartaDOS X itself and its utilities the changes had to be done with regard to the mechanism of density detection, which, until now, was based on the fact, that the size of the first sector is known, and that it is 128 bytes, and the size of the rest of the sectors can be determined from its (the first sector's) contents. The other problem is that a 512 BPS disk can be booted only as a hard drive partition, because the old XL OS is not able to set the sector size correctly either, and for a serial drive this means a checksum error and failure.

But, in our opinion, there are more advantages to this, outnumbering disadvantages. The first is that we are now compliant with the (real) standard disk devices used and produced around the world, where all the sectors are of the same size, and the smallest possible one is 512 bytes. Another one is having more boot region space (512 instead of 384 bytes) – creating new booter was possible at all thanks to that. Last but not least, the free space on the media is used more efficiently – although less than 1,5k is certainly a negligible loss on a harddisk.

Filesystem information

The filesystem information is basically the same as in the standard SpartaDOS format. The list of differences is below:

31 Physical sector size (1 byte): $80 – 128 bytes, $00 – 256 bytes, $01 – 512 bytes, other values are reserved. Generally, everything else than $80 is the high byte of the sector size measured in bytes, less 1.

32 Filesystem version number (1 byte): SpartaDOS 2.x, 3.x and SpartaDOS X 4.2x all set $20 here, which means 2.0. SpartaDOS X 4.40 has $21 here (version 2.1).

33-34 As of FS version 2.1: the „real” physical sector size value in bytes (2 bytes, low/high).

35-36 As of FS version 2.1: the number of sector entries per file map sector (2 bytes, low/high).

37 As of FS version 2.1: number of physical sectors per logical sector (cluster). Note that only one value – $01 – is supported at the moment.

These values are read-only in SpartaDOS X 4.40, it is not recommended to change them. Bytes 42-63 are reserved for future extensions and their values should not be altered for upward compatibility.

CAUTION: locations 33-37 have different meaning in the SpartaDOS 1.1 filesystem, and in the later versions of the filesystem, eventhough these bytes are not used, they retain values default for the 1.1. This is why the filesystem version number must be checked by the programmer before these locations are used in the program!

Directory structure

The directory structure is identical to the one found in the 2.0 filesystem. The only difference is that the first entry (the header) of the main directory has a valid time stamp: it is the date and time, when the filesystem was last built on that disk.

Direct disk access

Some programs need direct sector access to the disk bypassing the DOS – e.g. sector copiers – and as it has already been said before, since the *DD 512* density was introduced, you cannot hope anymore, that the first sector size is always 128 bytes. To determine the current disk configuration, where one can judge on the first sector size from, the READ PERCOM command should be used.

The program printed below is an example of a subroutine, which returns the information about the size of the sector number 1 in AX (low/high) for the drive number ($01-$0F) specified in accumulator:

```
ddevic     = $0300
dunit      = $0301
dcmnd      = $0302
dstats     = $0303
dbufa      = $0304
dtimlo     = $0306
dbyt       = $0308
daux1      = $030a
daux2      = $030b
jsioint    = $e459
buffer     = $0400     ;cassette buffer

getbootsize
        sta dunit
```

```
        lda #$31
        sta ddevic
        lda #'N     ;READ PERCOM
        sta dcmnd
        lda #$40
        sta dstats
        lda #<buffer
        sta dbufa
        lda #>buffer
        sta dbufa+1
        lda #$07
        sta dtimlo
;amount of data: 12 bytes
        lda #$0c
        sta dbyt
        lda #$00
        sta dbyt+1
;this shold be zeroed because of
;some floppy turbo systems (e.g.
;Top Drive, TOMS Turbo)
        sta daux1
        sta daux2
        jsr jsioint
        bpl success
;error 139 means that the drive
;does not know READ PERCOM cmd
;so it can only do 128 BPS
;(it is an Atari 810 or 1050)
        cpy #139
        beq a810
;any other error is just an error
        cpy #$00
        rts
;unmodified 810 or 1050,
;128 bytes per sector
a810    lda #$80
        ldx #$00
        ldy #$01
        rts
success
;low byte of the sector size
        lda buffer+7
;high byte of the sector size
        ldx buffer+6
;if the BPS < 512, return 128
        cpx #$02
        bcc a810
;or the returned value otherwise
        ldy #$01
        rts
```

PERCOM extensions

SpartaDOS X 4.40 recognizes an extension to the PERCOM standard. In the 5th byte of the PERCOM block (PERCOM+5) the previously unused 3rd bit (i.e. +$08) has now meaning as follows: when set (1), it means, that the disk does not have sides nor heads, thus the 4th byte of the PERCOM block (PERCOM+4) does not carry the number of them, but in its stead it contains the third byte of the number of sectors per track. Otherwise, when this bit is 0, the value of that byte should be ignored for hard disks, i.e. when the number of tracks is 1 (some hard drives tend to return the number of their physical heads here).

One of the less desired features of the new SpartaDOS X release is that it requires more memory. Because of that it is recommended to use the DOS on machines equipped with at least 128k of RAM, and to configure it so that the extended memory is used by the DOS (USE BANKED in the CONFIG.SYS file).

Generally, the low free memory pointer (MEMLO) should never go higher than $2000 for most programs to be executed. When its value is bigger, the „Memory conflict" error may occur much more often. The MEMLO value should be taken into account when installing fancy drivers, especially if the DOS is configured to run under the OS ROM (USE OSRAM) and the buffers are located in the main memory.

If the computer does not have more than 64k of RAM, or the RAM extension is about to be used in a different way, the best solution is to put DOS buffers under the OS ROM (USE OSRAM / DEVICE SPARTA OSRAM in CONFIG.SYS). In such a case the MEMLO value remains relatively low (around $1100 with SPARTA.SYS and SIOOLD.SYS), so you can load more drivers.

Whilst estimating the MEMLO value you should take into account the fact, that this pointer is raised by the X.COM program, which in turn is necessary to run most application programs. Therefore it is a good practice to do the command LOAD X.COM first, and then check the MEMLO state (with MEM command).

Character Sets

When the SpartaDOS X is configured to use the RAM under the OS ROM for buffers (USE OSRAM / DEVICE SPARTA OSRAM in CONFIG.SYS), a problem may occur with the international character set (CHARSET 2). A copy of the character set is made in the RAM shadowing the OS ROM to avoid ugly screen effects when the memory is being banked, the same memory area, however, is allocated for DOS buffers. When the number of the buffers exceeds certain limit (i.e. when they are more than 6), the font gets overwritten and the screen effects named above do appear.

The solution is to keep the number of the buffers equal or lower than 6 (the default is 4) in this configuration. It should be said again, that these remarks only apply, when one uses the CHARSET2 **and** the DOS buffers are under the ROM.

The CONFIG.SYS file

The default CONFIG.SYS file is to be found now on the CAR: device (CAR:CONFIG.SYS). It is read from there, when no user-defined

CONFIG.SYS is found on the boot disk. The environment variables CAR, BASIC and TEMP are defined by the RAMDISK.SYS driver, but only if the user did not define them before.

The COMSPEC variable is not defined, but its meaning and function remain the same, as in earlier versions of the DOS (it contains the pathname of the shell).

Any line beginning with a semicolon (;) is considered a comment and ignored.

The MERGE keyword

Apart from the usual configuration commands, i.e. USE, SET and DEVICE, there is a new one: MERGE. Its use is optional, but *when it is used, it must be the last keyword in the current configuration file*, because it aborts its processing and merges another one. This allows you to form a chain of text files to be processed at system startup, for example:

```
USE BANKED
MERGE DEFAULTS
```

The system will configure memory and then load a file named DEFAULTS.CFG. That file should contain actual configuration commands – and also may contain another MERGE at the end, if it is necessary to merge a portion from yet another file.

This feature is useful in conjunction with the multiple configuration files managed by the Config Selector (below): such configuration files, when they differ only at the beginning, may define differences in small number of commands as shown in the above example, and then merge the rest of the contents from common source. This allows to keep several configurations consistent by editing only a single text file.

The limitation to MERGE is that the merged file must be located in the same directory as the file, that merges. It also cannot be used before the USE keyword occurs in the stream of configuration commands. The file name extension (*.CFG) is optional.

Config Selector

SpartaDOS X v. 4.40 offers a built-in config selector. This feature allows the user to store several alternative CONFIG.SYS files on the disk, and decide at boot time, which one is to be used instead of the default one.

At boot time the DOS searches the main directory of the boot disk for a subdirectory named SPARTA.DOS. When it is found, and contains *.CFG files, a menu is displayed where each of the *.CFG files (up to

nine) is assigned a number. Hitting the appropriate key chooses the corresponding file to be used to configure the DOS. If you press any other key (Esc, Space, Return) or wait few seconds, the DOS closes the menu and continues with normal procedure.

For this to work the boot disk must be in SpartaDOS format.

The Drivers

There is a number of changes in SpartaDOS X system drivers, there are some new drivers as well. These news will be discussed in the following sections.

1) FILESYSTEM DRIVERS

SPARTA.SYS Driver

The structure for a single file (as in *nfiles*) has grown up from 35 to 40 bytes. The rest of the parameters did not change (minimum 2, maximum 16, 5 by default).

Because of the added support for 512-byte sectors the buffers (as in *nbufs*) have been enlarged to 512 bytes each. The maximum number of them has simultaneously been reduced to eight in some configurations (USE NONE and USE OSRAM with the buffers kept under the OS ROM). When the main or banked memory is allocated for buffers, the maximum is 16. You cannot declare fewer than 3 buffers, and the default number is 4.

It should be kept in mind, that 16 buffers, 512 bytes each, take twice as much memory as the same number of 256-byte buffers. If the DOS is told to USE BANKED, and 16 buffers are declared, it is quite likely that the extended memory bank the system uses will get completely filled up – in this case any drivers loaded afterwards will occupy the main memory, and the MEMLO will get raised. A good practice then is to never declare more than 12 buffers, unless a bigger number of them is *really* required.

The SpartaDOS X Reference Manual in this section is not quite accurate explaining, what the „buffers" are. Unlike the other DOS-es, where a buffer is usually assigned in a fixed manner to an open file, the SpartaDOS X buffering mechanism is a sort of a sector cache. This cache is maintained by the SPARTA.SYS driver to keep last accessed sectors, regardless of their type, i.e. whether these are boot sectors, bitmap sectors, file map sectors, data sectors or whatever. The greater the number of buffers, the less often the DOS is forced to re-read required data from the actual media. So, decreasing the number of buffers is unlikely to cause errors, but it certainly will make the filesystem work slower.

2) BLOCK I/O DRIVERS

SIO.SYS Driver

The SIO.SYS driver has been greatly improved over the earlier versions. First of all, an Ultra Speed drive is asked first, what serial speed it prefers (the old SIO was fixed at 52 kbps). Next, once the US mode was enabled, the SIO does not fall back to 19200 bps so easily, when an error occurs – so the TOMS drives can spread invalid responses around as they used to do, and the transmission still remains at the turbo baudrate. If the speed selected automatically turns over to be invalid anyway, you can still change it by hand using SIOSET command (see there).

Additionally there is a built-in mechanism of „mapping" disks, accessible by the MAP command (see there).

When someone does not need all that, and would like to get maximum free memory instead, there is the old-fashioned SIO driver on the CAR: device named SIOOLD.SYS. The only change that it underwent, was to make it work with 512-byte sector devices.

SIO2.SYS Driver

This is a lightweight driver for disk devices (an alternative for SIO.SYS). While not implementing custom communication routines, it uses the SIO subsystem of the computer operating system instead. Transmission parameters depend on the OS capabilities: the original Atari OS is rather poor in this area, while some third-party OSes support turbo transmission or uncommon devices.

The SIO2.SYS cannot be used, when the DOS is configured to USE OSRAM mode.

CA2001.SYS Driver

Purpose: the fast serial I/O program for California Access 2001 floppy drive.

Syntax: DEVICE CA2001 d:

Type: external – on device CAR:

The driver does not allocate memory, it only tries to enable fast serial protocol (38400 bps) for the specified CA-2001 drive. The drive remains engaged as long as it is powered on.

RAMDISK.SYS Driver

The ramdisk is installed by default as the drive number 15 (O:). Additionally, if the users did not define them otherwise, the RAMDISK.SYS driver defines environment variables $BASIC, $CAR

and $TEMP so that they point to appropriate filenames and its drive number.

PBI.SYS Driver

Purpose: additional support for PBI devices.
Syntax: DEVICE PBI
Type: external – on device CAR:

The PBI.SYS driver improves support for the parallel bus devices, such as hard disks. Some programs may fail to load from such a device because they require the data to be loaded to the memory, that shadows the PBI ROM area, and the parallel device driver residing in that ROM is naturally not able to write data under itself. The PBI.SYS solves this (rare) problem. It also somehow speeds up the transfers on systems, where there is only one PBI device present.

Caution: MAP and SIOSET commands doesn't handle PBI bus devices with the driver installed.

3) TIMEKEEPING DRIVERS

ARCCLOCK.SYS Driver

Purpose: the ARC clock driver.
Syntax: DEVICE ARCCLOCK
Type: external – on device CAR:

This is the driver for battery-backed real time clock called ARC (Atari Real Clock).

Z.SYS Driver

Purpose: SpartaDOS 3.2-compatible Z: device.
Syntax: DEVICE Z [/IS]
Type: external – on device CAR:

A „Z:” device is created in the OS handler table, which offers a simple interface to SpartaDOS timekeeping functions, accessible e.g. from a BASIC program. The device is compatible with the driver existing for SpartaDOS 3.2 (ZHAND.COM there), so the old software using that should have no problems accessing the desired functions under SpartaDOS X anymore.

The Z.SYS features four internal functions selected with BASIC XIO instructions (or appropriate OS directives):

1) XIO 33: read time, unformatted.
2) XIO 35: read date, unformatted
3) XIO 36: set time
4) XIO 37: set date

The correct procedure to read the time is as follows:

```
10 OPEN#1,4,0,"Z:":REMopen for reads
15 REM select reading time
20 XIO33,#1,4,0,"Z:"
25 REM get the clock state
30 GET #1,H:GET #1,M:GET #1,S
35 CLOSE #1
```

Setting time:

```
10 OPEN#1,8,0,"Z:":REMopen for writes
15 REM select setting time
20 XIO 36,#1,8,0,"Z:"
25 REM set the clock
30 PUT #1,H:PUT #1,M:PUT #1,S
35 CLOSE #1
```

The procedure to get and set the date is identical, you just have to change the XIO function codes to 35 and 37 respectively.

An attempt to read or write more than 3 bytes causes the error 136 (EOF) to occur. To reset the read/write pointer you should close and reopen the device, or call the appropriate XIO function again.

The functions setting the clock are disabled by default, attempts to write to the „Z:" device will cause the error 139 (NAK) to occur. Installing the driver with the /I switch (as in *ignore*) changes the returned status to $01 (success), but nothing else is done. To enable these functions you should load the driver with /S switch (as in *set*) given as a parameter.

The Z.SYS can only handle one I/O stream – an attempt to open more of them simultaneously will return error number 161 (Too many channels open).

Loading TD.COM enables more functions:

5) XIO 38: TD display line enable (TD ON)
6) XIO 39: TD display line disable (TD OFF)
7) XIO 34: read date, formatted
8) XIO 32: read time, formatted

These functions will only work, when TD.COM was loaded – or error 139 (NAK) will be returned otherwise. Example:

```
 10 DIM TIME$(13):REM reserve at least 13
bytes
 15 OPEN#1,4,0,"Z:":REMopen for reads
 20 REM read formatted time
```

```
25 XIO 32,#1,4,0,"Z:"
30 INPUT #1;TIME$
35 PRINT TIME$
40 CLOSE #1
```

Z.SYS requires a hardware clock driver to be loaded first: CLOCK.SYS, ARCCLOCK.SYS, JIFFY.SYS or any other compatible driver.

4) SCREEN DRIVERS

XEP80.SYS Driver

Purpose: XEP80 device handler.
Syntax: DEVICE XEP80 [1|2] [/P|/N]
Type: external – on device CAR:

The XEP80 can now be connected to either one of the joystick ports. The first parameter is the port number to be used (1 or 2, default is 2).

Earlier versions of XEP80.SYS driver contained a bug, which prevented the driver from working on PAL computers. The current version recognizes such machines correctly. Additionally, you can force either display mode by adding switches to the XEP80.SYS command line: „/P" for PAL or „/N" for NTSC.

QUICKED.SYS Driver

Purpose: screen accelerator.
Syntax: DEVICE QUICKED
Type: external – on device CAR:

QUICKED.SYS is a software screen accelerator. It installs into the CON: (DOS) and E: (OS) devices, speeding up the GRAPHICS 0 console operation up to four times.

CON64.SYS Driver

Purpose: 64-column screen console.
Syntax: DEVICE CON64
Type: external – on device CAR:

This is an experimental driver, that installs into the CON: and E: devices, and emulates a 64x24 text console in software using the 320x192 graphic display. After installing, it defaults to the standard 40x24 text mode. While in the Command Processor, you can enable the 64-column text mode by typing „CON64 ON" at the DOS prompt, and then hitting

the Return key, and later disable it similarly with „CON64 OFF”.

The 64-column console is not very useful for Command Processor operations. First, the screen is, in fact, in GRAPHICS 8, the 320x192 bitmap mode, and occupies nearly 8k of the main memory, having set the MEMTOP value at $8035. Few programs are actually happy with this. Second, not every SpartaDOS X utility can cope with the screen configured so – for example, MENU.COM and ED.COM fail miserably. Some of these problems may of course be fixed in future releases of the DOS.

The driver, however, may quite happily be used in BASIC and in your own programs. It adds two functions to the OS' „E:” device:

1) XIO 64: enable and disable the 64-column mode
2) XIO 65: check the presence of the CON64.SYS driver in the memory

When the driver is loaded, regardless of whether the 64-column mode is enabled or not, XIO 65,#chn,12,0,”E:” should execute without errors – or it should return status code 146 (Function not implemented) otherwise. „chn” is the number of the channel open for the console device, usually 0 (referenced as „16” in Atari BASIC). The other values should be as they were given to the OPEN call – in assembly you can omit setting them. GRAPHICS 0 corresponds to OPEN #0,12,0,”E:”. The call returns the current status of the driver in ICAX5 ($034E+IOCB*16): 128 if enabled, 0 otherwise.

If XIO 65 went well, you can use XIO 64,#chn,12,0+m,”E:” to switch to the 64-column text mode and back. If the „m” parameter is 128, the 64-column text mode will be enabled, and when it is 0, it will be disabled. The call returns the previous value of the parameter „m” in ICAX3 ($034C+IOCB*16).

The 64-column screen console functions identically to the normal 40-column one, just the logical line is longer: while it still can consist maximum of three physical screen lines, a physical line, however, consists now of 64 characters instead of 40 – and that sums up to 192 characters per logical line.

The driver also installs into the S: device. Under the control of the CON64 driver, there is no traditional form of text window anymore in any display mode – the 64-column console driver is not able to setup or handle such a window. The text, however, can be less or more freely mixed with graphics. The operation of the GRAPHICS 0, 8 and 24 modes under the control of the driver is as follows:

1) GRAPHICS 0: this is the 64x24 text mode. In this mode, the BASIC's POSITION keyword is effective on the *text cursor only*. You will probably be able to draw points or lines on it using the OS' PLOT and DRAWTO functions, but it is not recommended, since both S: the

display driver and E: the console driver share the same screen coordinates, and so it is quite likely that an attempt to draw anything via the former will result in position range errors reported by the latter.

2) GRAPHICS 24: this is the 320x192 bitmap mode. In this mode the BASIC's POSITION keyword is effective on the *graphic cursor only*. You will probably be able to print text on it using appropriate commands of the E: device, but it is not recommended for same reason as above.

3) GRAPHICS 8: this is the 320x192 bitmap mode with text window. In this mode, just as in the GRAPHICS 24, the BASIC's POSITION keyword is effective on the *graphic cursor only*. The text cursor position can be controlled through OS variables TXTCOL ($0291) and TXTROW ($0290), for the *x* and *y* coordinates respectively. In this mode you can safely both print text and draw graphics, because the screen driver maintains two separate sets of coordinates for the text and graphic cursors.

Note that the „text window" is not limited to the bottom three lines of the screen, but it covers the entire graphic display. This has some side effects, such as clearing the text screen also clears graphics, and vice versa.

The CON64.SYS driver requires an XL/XE computer equipped with a 130XE-compatible memory extension.

CON80.SYS Driver

Purpose: 80-column screen console.
Syntax: DEVICE CON80
Type: external – on device CAR:

This is an experimental driver, that installs into the CON: and E: devices, and emulates an 80x24 text console in software using the 320x192 graphic display. Its operation is very similar to the CON64.SYS driver described above, so this section will only discuss differences between them.

After installing, the system defaults to the standard 40x24 text mode. While in the Command Processor, you can enable the 80-column text mode by typing „CON80 ON" at the DOS prompt, and then hitting the Return key, and later disable it similarly with „CON80 OFF".

The driver adds two functions to the OS' „E:" device:

1) XIO 80: enable and disable the 80-column mode
2) XIO 81: check the presence of the CON80.SYS driver in the memory

These work identically to the XIO 64 and XIO 65, respectively, provided by the CON64.SYS driver. Also the general operation is similar,

the only difference being, that under the control of the CON80.SYS, the logical line of the screen editor can be as long as 240 characters. Such long lines, however, can confuse BASIC interpreters, so it is not recommended to exceed 192 characters per line while typing in a BASIC program.

Emulating 80-column text on a 320-pixel wide display requires much less calculations than for 64 columns. This is why the CON80.SYS is shorter than its 64-column brother, you can also find it being slightly faster.

The CON80.SYS driver requires an XL/XE computer equipped with a 130XE-compatible memory extension.

5) KEYBOARD DRIVERS

CAD.SYS Driver

Purpose: „soft reset" procedure.
Syntax: DEVICE CAD keycode repeat ON|OFF
Type: external – on device CAR:

Some programs, such as Disk Communicator 3, do not offer an option allowing the user to return to the DOS. The CAD.SYS solves this problem: when it is necessary to „kill" a running program, it is enough to press a defined key combination, and the system returns to the Command Processor, closing all the files and cleaning up everything it can.

The *keycode* parameter is a keyboard scancode of the key combination, which activates the „soft reset". The recommended values are: $E7 (Ctrl/Shift/Inverse) or $CC (Ctrl/Shift/Return).

The *repeat* parameter defines, how many times in a row the key combination should be pressed to activate the procedure. A zero means 256 times.

The last parameter decides whether the keyboard should generate upper (ON) or lower (OFF) case letters by default.

6) APPLICATION DRIVERS

RUNEXT.SYS Driver

Purpose: file association support.
Syntax: DEVICE RUNEXT [d:][path][filename.ext]
Type: external – on device CAR:

RUNEXT.SYS is an extension to the Command Processor, that allows to define associations between data types and application programs. For example, if the *.ARC files are associated this way with the ARC.COM

archiver, and the user types in an *.ARC filename at the command prompt, the Command Processor can automatically execute the archiver and hand over the specified filename along with predefined arguments to it.

The optional argument to the RUNEXT.SYS is a pathname to its configuration file. When none is given, the CAR:RUNEXT.CFG will be used.

The config file consists of lines defining one association each, and of comments (a comment has a semicolon or an asterisk at the beginning). The format of a line defining an association is the following:

EXT,PROGRAM [,PARAMETERS]

where:

EXT – three-letter filename extension (file type) to be associated.

PROGRAM – file name (with optional path) of the executable we associate with the file type above.

PARAMETERS – optional arguments to be handed over to the program; if nothing is defined here, the only argument passed to the program will be the data file name; if the file name has to be given at certain point of the command passed, we mark this place with a percent (%) character.

Example:

ARC,CAR:ARC.COM,L %

This is an association for ARC archives. Such files will be opened using CAR:ARC.COM. The first parameter handed over to it will be „L", the second – the archive file name. As a result, typing in an archive name at the DOS prompt, for instance:

```
D1:ARCHIVE.ARC
```

and hitting the Return key causes the archive's contents to be listed on the screen.

Entering the „+" sign at the beginning of a command causes bypassing the RUNEXT.SYS while executing the command.

COMEXE.SYS Driver

Purpose: automatic cartridge management when launching programs.

Syntax: DEVICE COMEXE

Type: external – on device CAR:

COMEXE.SYS is a system extension, that distinguishes between *.COM and *.EXE type binaries causing the DOS to load them in slightly

different manner. The *.COM files are considered external *commands* and simply searched for and loaded as before; now the *.EXE files are searched for and loaded, too, but before that the SpartaDOS I/O library module is automatically switched off releasing the cartridge area at $A000-$BFFF. In other words, if a binary has an *.EXE extension, it is a signal for the DOS, that it should be executed using X.COM – the system can now do it for you automatically, you do not even have to care about typing in the extension at the DOS prompt.

Entering the „+" sign at the beginning of a command causes bypassing the COMEXE.SYS while executing the command.

7) OTHERS

Program INIDOS.SYS

Purpose: re-starting SpartaDOS X.
Syntax: none.
Type: external – on device CAR:

Executing the command **COLD /N** in the Command Processor, or causing a cold system restart while in BASIC or most application programs, deactivates the SpartaDOS X cartridge completely. The reactivation is usually not possible without switching the power off and back on – and this in turn, for example, invalidates ramdisks.

This problem can be solved using the INIDOS.SYS program. Copy it onto a disk, where the computer can be booted from, and then type in the command **BOOT INIDOS.SYS**. When the SpartaDOS X is about to be reactivated after COLD /N, you only need to insert this disk in the boot drive and then reboot the system (without switching the power off).

Chapter 9 – Miscellaneous notes

Using BASIC XE Extensions

Due to the shortage of free RAM the DOS now uses the area at $D800-$DFFF to keep its internal structures while in USE OSRAM mode. BASIC XE Extensions load into the same place, so the statement expressed in the original SpartaDOS X (4.20) Reference Manual, that you can use BASIC XE Extensions in OSRAM mode, is no longer valid. If you want to load the BASIC XE Extensions, the DOS must be configured in BANKED mode and the computer has to have more than 128k RAM.

DiskRx

The SpartaDOS sector editor DiskRx does not recognize the new *DD 512* density, thus cannot be used to edit disks formatted in this way. Since the source code is not available, there was no other solution for that, than to write a suitable disk editor from scratch. The program has been written, it is called *Eddy*, and can be downloaded from *http://drac030.krap.pl/en-sparta-pliki.php*

Another problem with the DiskRx is that the program checks the filesystem type on the disk, where it is loaded from, and refuses to work, if it is not a SpartaDOS filesystem. The extended SpartaDOS FS built on a 512 BPS disk is not recognized as correct by this procedure, so DiskRx 1.9 cannot be loaded from such a disk. There is a patched version marked *1.9a*, where this problem is fixed. It still, however, cannot edit the *DD 512* disks.

CleanUp

The CleanUp program does not have a clue about the newly introduced *DD 512* density and thus, cannot be used to check and repair the filesystem built on such a disk. A program is planned to address this issue.

Appendix A

Error messages – as of SpartaDOS X v. 4.40 there are changes as follows:

148 – the message is changed from „Unrecognized diskette format" to „Unknown filesystem".

166 (Range error) – in a file operation this means: while reading, an attempt to read data or seek past the end of the file; while writing, the file exceeded its size limit (the limits are: 16 MB for a regular file and 32 kB for a directory). Generally: a parameter for the operation is beyond the allowed limit.

169 „Directory full" – new file cannot be created, because there is no space left in the directory to store its name. A directory may contain maximum 1423 entries for user files and directories. In earlier SpartaDOS X versions an attempt to exceed this limit caused the error 162 and the message „Disk full" used to appear, not quite accurate, since the files still can probably be saved to the disk, just not in this particular directory.

176 (Access denied) – the first *block_io* function called for a disk was not function 4 (*bio_rdsys*), the disk drive number specified equals 0 or is greater than 15, or an invalid function number was specified.

179 „Memory conflict" – an attempt to load a program, which overlaps the DOS kernel or the I/O library area. It often means, that the program has to be executed using X.COM.

181 „Filesystem corrupt" – the DOS cannot do the requested operation, because the filesystem structure on the disk is damaged.