Atari LiteDOS-SE

(for Atari 400/800 XL/XE Homecomputers)
Found on www.mr-atari.com

What is DOS:
DOS is the abbreviation for Disk Operating System and is basically an extension to the build in OS (operating system). Although Atari's build-in OS has the ability to do DIO (disk-IO) it has not the functionality of working with files in such a way the user can easily store and retrieve them from disk. Here DOS comes to the rescue.

I wrote this alternative DOS version from scratch for the fun of coding in 6502-assembler, but also to get a better knowledge how DOS and the OS work and interact. After numerous years, I am still learning and coding to enhance the performance and or remove quirky bugs. So without feedback from you guys, the users, there will be no nuts, no glory, no evolution.

The main characteristics of LiteDOS are:
-EXTREMELY SMALL, it's size is only 2k.
-FAST, it boots in a few seconds.
-SIMPLE, doing dir and loading files wasn't this easy.
-COMPATIBLE, it reads Atari DOS type 2 and clones like MyDOS.
-ALL ROUND, supports all drives and partitions up to 16mb.
-FLEXIBLE, choose between more files or more storage.
-19k2 BlueTooth support (hold SHIFT during 1st boot-sector).
-FREE, LiteDOS is free to download/distribute and use.

To the best of my knowledge, this is the smallest and easiest DOS version around. Giving programmers about 4k more RAM to use.

Please refer to the technical section if you have questions about why and how LiteDOS works.

Using LiteDOS:

Boot your system using the LiteDOS diskette, turn on the diskdrive first, then your computer. A welcome message will appear, showing the LiteDOS version while loading. To enable BlueTooth support, hold SHIFT during the 1st boot-sector, a message will appear that BT is patched. After booting, LiteDOS will pass control to your computer. Depending on configuration, a program language cartridges or the command-line of LiteDOS.

Your computer now has a device called D:, this is your diskdrive handle. Diskdrives can range from D1: to D8:, and when installed, D9: is your ramdisk.

LiteDOS works with file-names in the 8.3 format.
That means a 1-8 character filename plus an optional 3 character extension, separated with a dot. LiteDOS also support filenames exactly 11 characters long padded with space, that matches the 8.3 name without the dot.

Extensions are normally used to remember the file type.
Some examples are:
BAS Basic file
ASM Assembler file
LST Listing
XEX Executable
CEX Compressed executable
OBJ Objectcode
DRV Driver installation file

Using a program language cartridge (BASIC):
To store your program use SAVE or LIST and to retrieve it use LOAD or ENTER. Where LIST/ENTER use standard atascii files and SAVE/LOAD are language specific formats. Files in atascii are normally larger in size, but language independent.

Example(s)
LOAD"D:HUNT.BAS" will load the file HUNT.BAS
Since there is no drive number after D, LiteDOS will use drive #1
LOAD"D7:HUNT.BAS" will load the file from drive #7
SAVE"D9:TEMP" will save your work to the ramdisk with the name TEMP
LIST"D:HUNT.LST" will save in atascii format

Typing DOS will get you to the LiteDOS command-line, where 1 though 9 gives you a directory content and <reset> or G E474 will bring you back to BASIC.

Bootstrapping:
During boot, you can automatically load/execute a maximum of 10 files. These files must have the extension AU0 though AU9. AU0 is loaded first and continues until a file is not found or other load-error occurs. Bootstrapping is also cancelled when a program returns with the CPU-status set negative. Indicating a loading/init failure.

LiteDOS command-line:

The command-line of LiteDOS has basic functions for browsing disks, loading files and execute loaded/resident code. If you want to do disk maintenance, please read the chapter about DUP.

In the command-line you only see a inverse number on the screen. This number is the current drive LiteDOS is talking to.

When you have booted with a language, like BASIC, type DOS <return> to enter. Use RESET or G E474 to exit the command-line.

Directory (number/[device]/return):
1 through 9 - Show the directory of drive #1...9
You can add <space> filename or <space> wildcard to narrow browsing.
Like 1 *.BAS will list all files ending with .BAS on D1:
Optional you can browse another device if that device-driver supports the DIR-function.
Examples; 1H will do DIR on device H1: or 1N from your network device.
When sub-directories are encountered (MyDOS for example), all matching files are listed, but not the name of the directory.

Load (navigation/return):
After browsing, you can load the file by simply navigating to the line with the file-name and press return. Files must be machine language and in the correct format. This feature does not work on other devices then D.

Direct file loading:
When you know the name of the file and what device it's on, type that in.
D1:RWTEST.XEX will load (and execute) this file found on D1:
Wild cards are allowed, so D:RW*.* will probably load the same file.
N1:TNFS://SERVER/FILE.XEX will load FILE.XEX from the network device N:

Execute (go):
Type G xxxx return.
Were xxxx is a hexadecimal address.

Some addresses of resident code:

G E477	Coldstart
G E474	Warmstart
G E471	Selftest (XL/XE) or Memopad (400/800)

LiteDOS.DUP, the Disk Utility Package:

DUP is a program for maintaining your disks.

Loading DUP:
From the command-line, do DIR of the correct disk, navigate to the LiteDOS.DUP file and press return. If you know it's present in drive 1, type D <return>. If you know it is on D8, typing D8:*.DUP <return> might be faster. To enter the command-line, first type DOS.

About DUP:
Basic functions are copy, erase and rename.
Beside this, it offers output to screen or printer and write DUP to another disk.
Exit DUP will clear user memory or restarts with forced boot (B) if MEMLO was above $2000 when DUP was loaded.

The navigation-method is the easiest way to do maintenance. Simply do a DIR, navigate to the line with the filename you want to work on and type the command in front of it. The space between the command and the filename is a field for optional information. This optional information can be a drive number (1 thourgh 9) or device name (A through Z).

Beside the navigation-method, you can also type-in the command.
Unsupported commands will send out Error-168, all other errors are device specific, see your Atari/Device-manual for Error-code explanation.

Almost all commands will ask for conformation, so basically you can't do something bad. The screen will blink red/green here, press Y for yes. N for No, ESC for Escape and A for All.

I don't think DUP need much explanation, browsing is the same as in the command-line. All file manipulations are done from the current device.

Rename is a bit special, to execute correctly, put a comma and a new name behind the current filename (you can overwriting the file-length when navigating)

Some examples:
C3 will copy all files from current device to drive #3
C3 *.BAS will copy all files ending on .BAS to drive #3
CH HUNT.BAS will copy the file to H1:
CC HUNT.BAS to cassette, using short gaps (default), compatible with CLOAD
CC HUNT.BAS,L to cassette using the long gaps, compatible with LOAD"C:"
W3 will write DUP to drive #3
P1 HUNT.LST will send the file HUNT.LST to P1:
R3 SIJMEN.FIL,SANDRA.FIL will rename the file SIJMEN.FIL into SANDRA.FIL on D3:
(Do NOT use rename on other devices then D: !)

Drivers and programs that come with LiteDOS (.XEX or .DRV files):

Drivers are memory resident programs that support devices. They can be loaded when booting the system, using the bootstrap-method or loaded when you need them. Loading a driver when you need them, will erase user memory and restart the program you are running.

Loading a driver when you need them:
From the command-line, do DIR of your driver disk, navigate to the .DRV file and press return, or, type D:name.DRV and press return. To enter the command-line, first type DOS.

Bootstrap-methode:
When LiteDOS boots, it looks for files ending with AU0-AU9.
These files are seen as auto-execute files and loaded in sequence of their number. The bootstrap method stops when a driver/load-error occurs or when no more .AU# files are present. Then control is passed to the loaded software, language or cartridge. This way you can load drivers when renamed to .AU#

Some available drivers and other code on this LiteDOS disk:

LiteRD.DRV, Ramdisk-driver, installed as D9 for 64k 130XE memory. If you have no extra memory, it will look for available shadow RAM in BASIC and OS region. Ramdisk access is faster then using a disk, but remember that data is lost when you switch off. Loading LiteRD.DRV a second time gives you the option to format.

Lite850.DRV, this will download/install your R: driver from 850 or compatible hardware (like FujiNet).

LiteHDD.DRV, driver that does IO over an IDE port. Current driver supports MyIDE, SIDE and XL-CF hardware. It patches D4: to IDE-access. IDE-Storage is seen as one large disk (16MB), using raw-data, ignoring/no partitions. "LiteDOS style".

LiteREMU.XEX, Not a driver, but it is a resident program to support "soft cartridges

LiteHSIO.DRV, IO-patch to support high speed IO on enhanced drives running faster then 19k2. These devices must support the speedbyte-command.

LiteBAS.XEX (thanks to DMSC. This is for bootstrapping an Atari- BASIC program. This program runs the first .BAS file it finds. Make sure BASIC is present/enabled and your BASIC program is saved as .BAS.

LiteLZ4.XEX: a LiteDOS-plug-in to unpack LZ4-SE compressed files during binary loading. The de-compressor runs from stack ($100).

19k2 BlueTooth is user activated (SHIFT at boot) and runs from stack ($178)

Version 2 drivers (HHD2, HIO2), stored on stack (stack must be free).

Making new disks or partitions for LiteDOS (LiteINIT.XEX):

LiteDOS supports standard drives, percom-drives and partitions in SD/DD format.
Where SD has 128 bytes/sector and DD has 256 bytes/sector.

Introduction to disk sizes:
Other (confusing) terms are the size of diskettes SD/MD/DD/SS/DS/HD, where the number of bytes/sector do not always reflect the size of the disk….

Standard Atari diskettes:
SD-diskette is a 720 sector disk with 128 bytes/sector holding 90kB of data.
MD-diskette is a 1040 sector disk with 128 bytes/sector holding 130kB of data.
DD-diskette is a 720 sector disk with 256 bytes/sector holding 180kB of data.

These are all Single Sided (SS) diskettes, read/written one side.
All drives can read/write SD/SS, where as MD and DD are already special formats.
Since most drives have only one head and you could flip the diskette for using the other side.

Next format is DS or Double Sided, these drives have 2 heads and you can not flip the disk to be read in a single headed drive (since the data is stored anti clockwise).
The XF551 is an example of a double headed drive.
These DS disks are in DD density.
DSDD/(XF551)-diskette, 1440 sectors with 256 bytes/sector holding 360kB of data.

Some people have their XF551 enhanced with a 80 track (40 is standard) drive, doubling the capacity again. Mostly a 3.5” mechanic for 720k diskettes.
720k-diskette, 2880 sectors with 256 bytes/sector holding a whopping 720kB of data.

Beyond this point we mostly talk about partitions or other media types like IDE/CF.
Here LiteDOS support both SD and DD, although SD is highly unusual.
With a maximum addressability of 65535 sectors (16Mb/DD or 8Mb/SD)
LiteINIT also supports smaller variants, from 2048 sectors upwards, steps of x2.

Using LiteINIT
Once you have made all necessarily selections for the disk you want to make, you need to press Y to confirm. ESC or any question will bring you back to the beginning of the program. Pressing ESC here will exit the program.
Choose the default setting when you are not sure (pressing return).

*)
Formatting in MD is a special case, here LiteDOS maintains the 10-bit sector link if you want a bootable LiteDOS disk using the upper 16 sectors of the disk (1024-1040) for the system files. For compatibility when reading this disk with DOS 2.5 Selecting “No DOS” switches to MyDOS standard using a 16bit sector-link.

Technical section of LiteDOS-SE:

"Everything is connected, these are wise words, keeping things small…"

Disk layout:
LiteDOS uses clusters of sectors (instead of single sectors) to store data.
But has single sector read-access to support others to read LiteDOS disks.
For compatibility, LiteDOS also starts counting on a not existing sector zero.

Keeping track of free/used clusters is done in a bitmap table, this is on sector 360.
Sector 360 is NOT compatible with others for saving, although others should test if this disk is "theirs", do not save on LiteDOS disk with another DOS (!)

The size of the VTOC (Volume Table Of Content) are two clusters with a minimum size of 2 sectors, always starting on sector 360. Sector 360 is the bitmap sector, the rest of this double cluster is reserved for the directory, therefore the clustersize is limiting or expanding the number of files on this disk. For compatibility, LiteDOS stores 8 names in a directory-sector, although on DD there is room for 16 names. File-names start on sector 361 counting up and wraps-around inside this double cluster.

Sectors/Cluster	Max. size (k)	Max. sectors	Max. files	VTOC-area start sector	end sector
1 (min.)	256	1024	8	360	361
2	512	2048	24	360	363
4	1024	4096	56	360	367
8	2048	8192	120	352	367
16	4096	16384	248	352	383
32	8192	32768	504	320	383
64 (max.)	16384	65535	1016	256	383

Sector 360:
Offset $00, 1 byte, holds the LiteDOS-SE ID (first 2 bits "01"))/ClusterSize-1 (6 bit)
Offset $01, 2 bytes, the size of this disk/media in sectors.
Offset $03, 2 bytes, the amount of free sectors, called "free space" in LiteDOS.
Offset $05, 5 bytes, all zero.
Offset $0A, 118 (SD) or 128 (DD) byte, bitmap table, first bit stands for cluster 0. Therefore having a maximum of 1024 bits (allocating 1024 clusters) for saving. Bits are set to zero when a cluster is occupied and set to 1 when free. The first 3 sectors (boot-sectors) are not available for LiteDOS and 2 clusters are used for the VTOC. In this unavailable space the bootloader is stored.

So, doing the math, covering a 720 sector disk, you need at least 1 sector/cluster. Doing so, giving 720 clusters of 1 sectors, and reserves 2 sectors (2 clusters) for the bitmap table and the directory, capable of holding 8 files.
Having 4 sectors/cluster would increase file capacity to 56 files (see table above).

LiteINIT will ask you how many files you want to store when using a diskette.

Directory layout (sector 361 and up):
The first directory sector starts at sector 361 and counts upwards, wraps inside the double cluster, until the maximum number of sectors is read. This is done for compatibility, so others can read the first part of the directory too. Each directory sector has 8 filename fields.

Each Filename fields are 16 byte long, coded like this:
Status: 1 byte, $00 empty, $80 deleted, $46/$42 LiteDOS file (16/10 bit link).
Length: 2 bytes (lo/hi), sectors occupied by this file.
Location: 2 bytes (lo/hi), starting sector of this file.
Filename: 8 bytes padded with space.
Extension: 3 bytes padded with space.

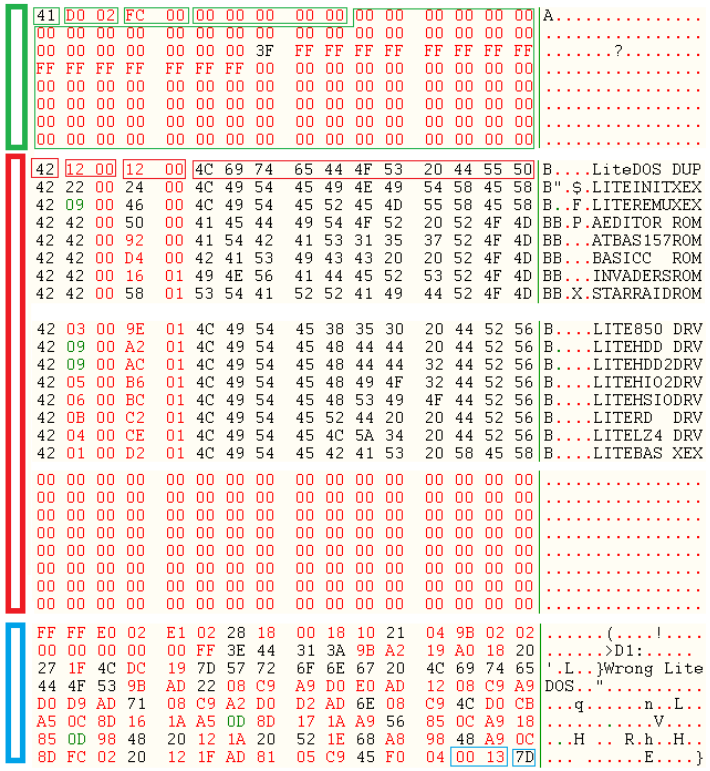
File layout:
Files are stored using a first sector (stored in the directory). Every data-sector has information about the next sector, file-number and bytes used in this sector.
This information is stored in the last 3 bytes.
Full data sectors therefore hold 125 or 253 bytes of data.

The last byte in a sector is the number of bytes stored on this sector.
The 2 bytes below is the sector-link.
On disks larger then 1024 sectors this is a 16 bit link.
Smaller disks have a 10bit link, the upper 6 bit hold the relative file-number (0 to 64) in the directory. The last sector in a file have a sector-link with the value nul.

Drivers and resident programs:
Drivers and resident programs should set there init routine in location DOSINI ($0C,2) for its own initialization, the driver should jump to the former values in DOSINI, hence to initialize the other driver(s).
When setting LOMEM, the driver should check if it's not already set higher then what the drivers needs.
If it is higher, do NOT set LOMEM. Same goes for HIMEM, if it is already set lower, do NOT set HIMEM.
During installation of the driver, the program should check COLDST ($244).
When non zero (power-up or bootstrapping is in progress) end the installation with a RTS and the installation status in the CPU. Positive means OK.Errors can be passed through using the Y-register. Like LDY #176, RTS otherwise do LDY #1 (OK)
If coldstart is not in progress, pass the control back to the calling software, using a cold-init to clear the program-area.
Drivers can be loaded at any time, it is not needed to load drivers up-on boot, you can load any driver any time you want.
As long as they support the LiteDOS guidelines set out above.

Mr.Atari 2022

Example of a SD diskette formatted with LiteDOS in 2 sectors/cluster (ID=$41)



Sector 360
LiteDOS-ID, bit 7/6 = 01, bit 5-0 = clustersize-1
$41 = Litedos, Clustersize=2, DirSize=4 sectors
D0 02 = $2D0, disksize = 720
FC 00 = $FC free sectors = 255
5x 00 = VTOC marker
Byte $0A start of bitmap-cluster-table.
1 bit equals 1 cluster, starting at cluster 0

Sector 361-363, Remaining 3 sectors for file-names
$42 file status, $4x = present, $x2 = 10 bit sector link
(00=free, 8x=deleted) $x6 = 16 bit sector link
12 00 = $0012 file lenght in sectors (not clusters)
12 00 = $0012 start sector
11 bytes, 8.3 character filename, all characters allowed.

Example of a data sector $12
00 13 = Link to next sector, 10 or 16 bit, sector $13 here.
If 10-bit, the upper 6 represent the file-ID
Here it is #00, so file #00 = LiteDOS.DUP
(example LITE850.DRV = file #08 or %001000xx
7D = number of bytes in this sector. Here 125 bytes.