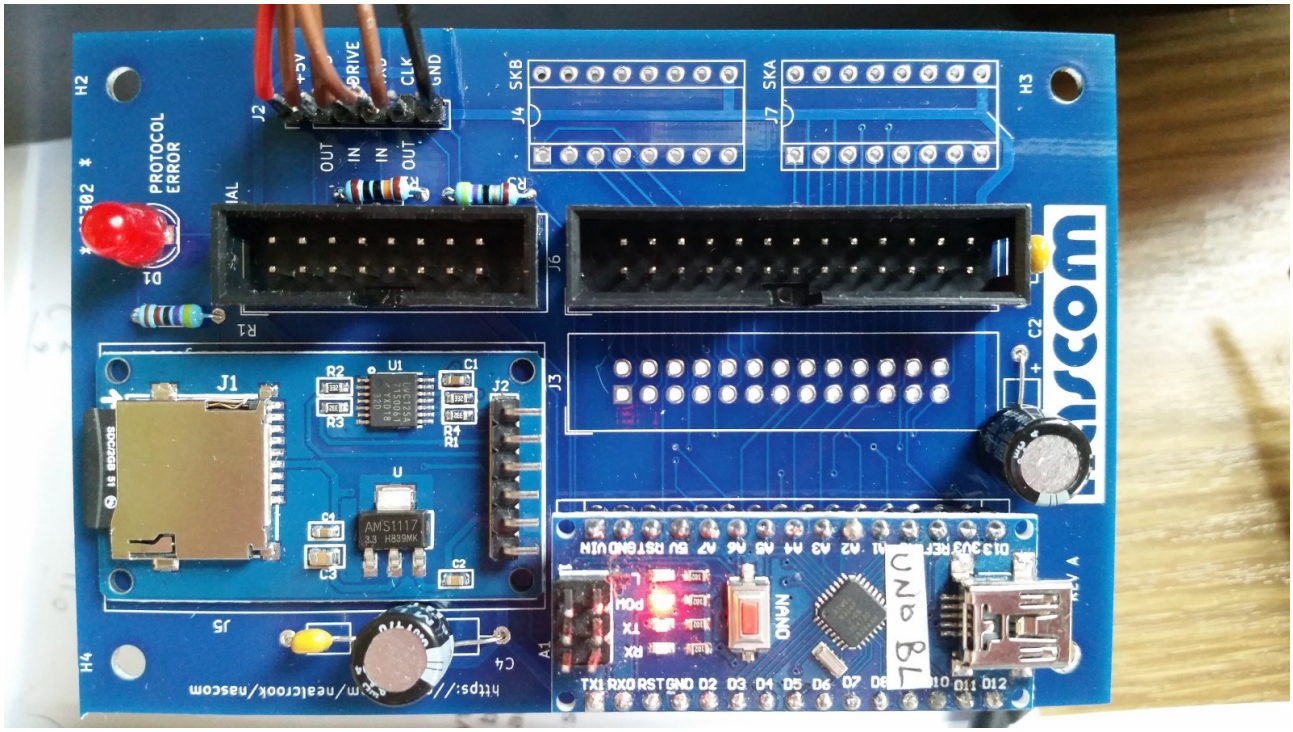
NASCOM SDcard Interface



Introduction

This add-on for the NASCOM 1/NASCOM 2 provides file-based storage on a (micro) SDcard. It provides 3 interfaces:

- Connected to the NASCOM serial interface it can be used as a replacement for a tape recorder, and interacts with the standard load/save commands from NAS-SYS, BASIC or any other application. This functionality is referred to as "**NAScas**".
- Connected to the NASCOM PIO and used in conjunction with a boot EPROM it can be used as a replacement for a floppy disk to run PolyDos or NAS-DOS, or as a fast way to load (eg) ROM images to RAM. This functionality is referred to as "**NASdsk**".
- Connected to a PC through the USB interface it can be used to monitor status and to transfer files between the PC and the SDcard. This allows cross-machine development without the need to constantly swap out the SDcard. This functionality is referred to as "**NASconsole**".

The hardware itself is referred to as "**nascom_sdcard**". It consists of an Arduino Nano, a micro-SD Daughtercard, an LED and a few connectors and passives mounted on a small PCB. The PCB is available from the author (~ £2.50) and the other parts are cheap and easily available (eg from EBAY, Banggood or Aliexpress). The prototype was built and debugged using an Arduino Uno and an Arduino prototyping "shield", but the Nano-based solution provides several advantages.

The SDcard stores files in FAT format. The PCB is connected directly to the NASCOM using existing connectors and draws power from the NASCOM.

The goals of the design are two-fold:

- To provide a mechanism for extracting data (ROMS, tapes, floppy disk contents) from NASCOM storage media
- To provide a "retro environment" in which a physical NASCOM machine can be used without recourse to tape recorders or floppy drives.

nascom_sdcard has been tested on a NASCOM 1 and on a NASCOM 2.

This document covers usage, preparing an SDcard, theory of operation, construction, programming and installation. Usage is at the front; the other sections are relegated to the back of the document because you may only care about them once.

Additional documentation (schematics, assembly drawings, photos, software, application examples and details on the internal protocols) can be found at https://github.com/nealcrook/nascom.

Errors and omissions

You may find some information here wrong or confusing. You may look on my github repository and not be able to find something that you are looking for: or it may seem wrong or out-of-date. Please report any such instances to the author.

Usage (Overview)

This document assumes that you *will* connect nascom_sdcard to the serial interface of your NASCOM and that you *may also* connect it to the PIO and that you *may also* connect it to a PC using the USB cable.

All three of these interfaces can be enabled simultaneously, but only one can be processing a command at any one time. For example, while you are copying a file to the SDcard using the NASconsole interface, you cannot use the NAScas command-line interface, nor will nascom_sdcard capture a file written using the NAS-SYS W command.

There are separate sections here to describe the usage of each interface.

Usage (NAScas)

When you power-up or reset nascom_sdcard, it sends "R<CR>" across the serial interface, which causes NAS-SYS to issue the "R" (read) command as though the “R<CR>” had been entered at the keyboard. NAScas detects the read command like this:

- It detects that the DRIVE signal has asserted
- It polls to see if data is received from the NASCOM (ie, a WRITE)
- After a certain time has elapsed and no data has been received, it infers that the NASCOM is expecting data (ie, a READ)

At this point, NAScas sends a byte stream to the NASCOM, in cassette R format. This byte stream is a bootstrap program called SERBOOT. The NASCOM loads it into memory at $0C80. After the

DRIVE signal negates, NAScas sends the text string "EC80<CR>". The NAS-SYS command loop always polls the serial/cassette interface at the same time as the keyboard, so this text string causes it to start execution of the SERBOOT program.

The SERBOOT source code is here: sdcard/host_programs/serboot.asm[1]

SERBOOT is tiny (110 bytes). It provides a prompt and command loop. Its function is to relay commands to NAScas and to report responses. The prompt looks like this:

```
NAScas>
```

You can issue a command, followed by <ENTER>. There are 2 ways to exit the command loop and return to NAS-SYS:

- terminate a command with a period, "." (followed by <ENTER>)
- enter a period, "." by itself on a line (followed by <ENTER>)

Usage Paradigm

Tape is a sequential access device so we need a paradigm that accommodates that, and that will work with all existing software. Unlike tape, which is a linear device, NAScas uses files. The basic usage model is this:

- Use the NAScas command to specify files that will be used for future "R"ead or "W"rite commands
- Exit NAScas
- (at some time in the future) issue "R" or "W" commands (or CLOAD/CSAVE from BASIC)

This idea of specifying file names in advance is analagous to "cueing" the tape by winding forward or back to the correct position.

There is some additional capability, which will be described along with the command-set.

Reloading/Relocating SERBOOT

Usually SERBOOT will stay in memory forever. You can always re-run it from NAS-SYS (EC80<ENTER>). If it gets lost or corrupted you need to reset nascom_sdcard and issue the R command again from NAS-SYS.

SERBOOT always loads and starts at $0C80 but the code itself is relocatable. The TO command can be used to move it to a new location:

```
NAScas> TO 1000<ENTER>
NAScas>
```

In this example, the code is relocated to address $1000. When the second prompt appears, the code is executing at the new address.

1 All references are paths relative to https://github.com/nealcrook/nascom

Formats and file-systems

NAScas supports 3 file-systems:

- Vdisk - a read-only file-system implemented in a binary blob that is stored on the SDcard. The binary blob is a PolyDos disk image; the format is documented in the PolyDos System Programmers Guide. File names use an 8.2 format: 1-8 characters before the dot, exactly 2 characters after the dot.
- Flash - a read-only file-system that uses the Flash memory on the Atmel processor of the Arduino. This stores the SERBOOT code, the DSKBOOT code (see page 12) and a few other programs (including ZEAP and Lollipop Lady Trainer). File names use PolyDos format (see above).
- SDcard - a read/write filesystem implemented on the SD card. File names use MSDOS 8.3 format: 1-8 characters before the dot, 1-3 characters after the dot.

NAScas supports these file formats:

- Files stored on the Vdisk and Flash file-systems are binary blobs with additional meta-data (load address, execution address) that is stored in the associated directory structure. When read, a file is converted to CAS format on-the-fly.
- Files stored on the SDcard FAT file-system can be text files (see the description of the TS command) or byte streams in CAS format. When NAScas writes to a file on the SDcard, it stores the exact byte stream from the NASCOM UART. When the file is subsequently read, it delivers that exact byte stream. Files that have been created elsewhere (e.g. for NASCOM emulators) in .CAS format can be loaded onto an SDcard and read as though they had been generated by the NASCOM.

Additional utilities exist (converters/polydos_vfs and converters/nascon) for creating/maintaining PolyDos disk images and for converting between CAS and other formats.

NAScas Commands

Only the first 2 characters of the command name are significant.

```
HELP - report help for all commands

INFO - report fw version and any other useful state info

. - quit the CLI (this is handled directly by SERBOOT with no communication with
the NAScas hardware)

TO xxxx - relocate SERBOOT to specified address.

DF - directory of Flash. This shows the programs SERBOOT.GO and DSKBOOT.GO and
any others that were configured for inclusion.

DV - directory of virtual disk. Output is displayed one screen at a time, with
an invitation to “Press [SPACE] to continue”. Output is aborted if any other key
is pressed. Error if no virtual disk mounted.

DS - directory of SDcard. Output is displayed one screen at a time, with an
invitation to “Press [SPACE] to continue”. Output is aborted if any other key is
pressed. Error if no SDcard present.
```

NEW - check for presence of SDcard and set working directory. Automatically performed at reset. Use this when you change the SDcard. If a directory named NASCOM exists, all file operations use this directory. Otherwise, all file operations use the root directory of the SDcard.

MO <file.xxx> - mount file as virtual disk. File can have any legal DOS 8.3 name. Error if no SDcard present. Error if illegal file name. Error if file not found.

RF <file.go> - cue file for reading from Flash. File can have any legal PolyDOS 8.2 name but all files in the Flash file-system have the .GO extension. Error if illegal file name. Error if file not found.

RV <file.xx> - cue file for reading from virtual disk. File can have any legal PolyDOS 8.2 name. Error if illegal file name. Error if no SDcard present. Error if no virtual disk mounted. Error if file not found.

RS <file.xxx> - cue file for reading from SDcard. File can have any legal DOS 8.3 name. Error if illegal file name. Error if no SDcard present. Error if file not found.

ES <file.xxxx> - erase file from SDcard. Error if illegal file name. Error if no SDcard present. Error if file not found. See section “MSDOS FAT (8.3) Filenames” below.

WS <file.xxxx> - cue file for writing to SDcard. Error if illegal file name. Error if no SDcard present. See section “MSDOS FAT (8.3) Filenames” below.

TS <file.xxx> - send file from SDcard as text. See section “The TS Command” below.

AUTOGO - after reset, any file that is read from Flash or Vdisk and has a known execution address is executed after loading (for files saved to SDcard use the NAS-SYS "G"enerate command to make them execute automaticaly after loading).

AUTOGO - toggle flag

AUTOGO 0 - clear flag

AUTOGO 1 - set flag

PAUSE n - after issuing a TS command the data stream will start straight away (will not wait for DRIVE light). This is the number of seconds to pause before the data stream starts (default 10).

NULLS n - after issuing a TS command this is the number of milli-seconds to wait after a line-end, to give BASIC time to catch up. It is not actually implemented by issuing NUL characters; it just uses a time delay (default 100).

The AI argument

The RS and WS commands allow an optional AI argument after the file name. This argument causes the last 2 digits of the file name to increment numerically after each read or write operation. INFO will show the next file names to be used.

Usually, if you issue RS repeatedly, you will always read the same file. Usually, if you issue WS repeatedly, you will always overwrite any existing file. Using AI allows multiple versions to be kept, which can be useful during program development.

The TS Command

The TS command can be used to send a text file to the NASCOM exactly as though you had typed it in by hand. The NASCOM uses CR (0x0d) line endings and ignores LF (0x0a), so your text file should use MSDOS line endings (Unix line endings will not work).

For example, you could grab a BASIC program from somewhere on the 'net and then deliver it to BASIC (which will tokenize it line by line and store it in memory) and then save it using CLOAD. The session to do this would look something like this:

```
NAScas> WS HANGMAN.CAS                  -- encoded file will be save here
NAScas> TS HANGMAN.BAS                  -- text file to be read into BASIC
NAScas> .                               -- leave the CLI

-- NAS-SYS 3 --
X0                                      -- allow long lines in BASIC
J                                       -- cold-start BASIC

Memory Size?

Microsoft BASIC

<program scrolls by line by line>       -- you have 10s from issuing the TS
                                           command to now

MONITOR                                 -- back to NAS-SYS
N                                       -- back to normal command handling
Z                                       -- back to BASIC
CSAVE "A"                               -- save encoded file as HANGMAN.CAS
```

MSDOS FAT (8.3) Filenames

To save space in the Arduino image, the FAT filesystem is configured to only support “8.3” filenames, not the “long filenames” associated with the vfat extension. 8.3 filenames must use upper case characters. If you have a file “UPPER.TXT” and a file “lower.txt” on the SDcard:

- DS will list them as UPPER.TXT and LOWER.TXT respectively.
- MO will allow you to mount them as UPPER.TXT and LOWER.TXT, respectively.
- ES will allow you to delete UPPER.TXT
- ES will report “file not found” if you attempt to specify LOWER.TXT or lower.txt
- RS will allow you to specify and read UPPER.TXT or LOWER.TXT (or upper.txt or lower.txt).
- WS will allow you to specify a filename in upper/lower/mixed case but the filename that’s used on the Sdcard will use upper case.

Probably, only the ES behaviour will cause confusion; the easiest way to avoid confusion is to stick to upper-case names. Beware: Windows sometimes causes extra confusion by displaying filenames in a case that does not reflect the way that they are actually stored!

Usage (NASconsole)

The Arduino on the nascom_sdcard hardware provides a USB connection that implements a virtual UART. When linked to a terminal emulator on a PC it reports status and debug messages. If the code is built with "`#define CONSOLE`" this interface also provides a restricted command interface.

The console interface protocol uses ASCII so that it can be mingled with status/debug messages. However, it is designed to be accessed though a special program on the PC, not simply a terminal emulator. For that reason there is almost zero error/sanity checking on arguments within the Arduino code: the implementation was focussed on small code size through the reuse of existing code.

On a PC, the PERL program called NASconsole is used to communicate across this interface.

NASconsole has a built-in "help" command that describes its command-set. The following commands are supported:

- Directory of SDcard
- Write: transfer file from host filesystem to SDcard
- Read: transfer file from SDcard to host filesystem
- Erase file on SDcard
- Read byte from Arduino EEPROM
- Write byte to Arduino EEPROM

The use-case is to allow cross-hosted development of NASCOM code on a PC without need to keep swapping the SDcard back and forth.

Starting NASconsole

NASconsole is started by specifying the USB port that is in use by nascom_sdcard. For example:

```
$ ./NASconsole /dev/ttyUSB0
```

When started, it responds with the prompt "`NASconsole>`". As well as waiting for and responding to commands it (asynchronously) reports status/debug messages.

When you start NASconsole, it resets the nascom_sdcard hardware; this is an Arduino design feature.

NASconsole commands

```
dir                     - Directory of SDcard
                          Error if SDcard not present

erase <filename>        - Erase specified file from SDcard.
                          Error if SDcard not present
                          Error if filename is not a MSDOS 8.3 name
```

```
                          Error if filename does not exist

write <filename>          - Write local file filename to SDcard. filename is
                          valid MSDOS 8.3 name.
                          Error if SDcard not present
                          Error if filename is not an MSDOS 8.3 name
                          Error if filename does not exist on local filesystem

write <lfilename> <filename> - Write local file lfilename to SDcard as the name
                          filename; filename is valid MSDOS 8.3 name
                          Error if SDcard not present
                          Error if filename is not a MSDOS 8.3 name
                          Error if lfilename does not exist on local filesystem

read <filename>           - Read file filename from SDcard and save locally;
                          filename is valid MSDOS 8.3 name
                          Error if SDcard not present
                          Error if filename is not a MSDOS 8.3 name
                          Error if filename does not exist on SDcard

read filename lfilename   - Read file filename from SDcard and save locally as
                          lfilename; filename is valid MSDOS 8.3 name
                          Error if SDcard not present
                          Error if filename is not a MSDOS 8.3 name
                          Error if filename does not exist on SDcard

help                      - Display summary of commands

quit                      - Exit NASconsole
```

For convenience, whatever is specified as "filename" is converted automatically to upper case; case is only important for "lfilename" (for write: if "lfilename" is not found, try to use the upper-case equivalent then the lower-case equivalent before giving up).

Usage (NASdsk with PolyDos)

PolyDos requires RAM up to $C000 and 2Kbytes of boot code; usually stored in EPROM at $D000[2]. Different versions of PolyDos (1, 2, 3, 4) support different disk controllers. The NASdsk version of PolyDos has been derived from PolyDos 2 and supports 4 virtual disk drives each of 350Kbyte capacity. PolyDos abstracts all hardware-related activity to the boot code and therefore there are no changes to the disk images.

Boot the system by executing from the EPROM start address ($D000 or $D800). The NASCOM screen should clear and display the prompt:

```
Boot which drive? _
```

In response, type 0. After a few moments you should see this exciting message:

```
PolyDos 2 (Version 2.1)
Copyright (C) 1982 by
PolyData microcenter ApS
$
```

If you see this message:

```
(Error 27)
– NAS-SYS 3 –
0
```

..it indicates a problem with your board or SDcard or SDcard contents.

You can find the PolyDos documentation here: PolyDos/doc/PolyDos2_doc.pdf

You can use any facility of PolyDos *except* the FORMAT or BACKUP commands (if you use those commands, they will attempt to access a real floppy disk controller).

Anders Hejlsberg, the author of PolyDos, emailed me as follows (29Apr2018): *"First, absolutely feel free to share anything you have related to PolyDos or any of the other software I wrote for the NASCOM 2. I'd be delighted to see any or all of it in the public domain."*

PolyDos Utilities

Drive 0 contains some new utilities. In all cases, the source code is here: sdcard/host_programs

- `SCRAPE – copy a PolyDos floppy disk image to SDcard`
- `SCRAPE5 – copy a CP/M floppy disk image to SDcard`
- `SETDRV – display and change virtual disks`
- `SDDIR – directory of SDcard`
- `CASDSK – Intercept NAS-SYS Read/Write commands`
- `SDOFF – Disable nascom_sdcard`

SCRAPE

Copy a complete floppy disk image to SDcard.

2 PolyDos calls are made by extending the NAS-SYS SCAL table and are therefore relocatable. The PolyDos ROM can be reassembled for "any" start address. In addition, it will run from RAM quite happily.

```
$ SCRAPE
```

Prompts you to "Insert disk then press ENTER, or SPACE to quit". After you press enter, each sector of the disk in turn is read, and the sectors are written to a file on the SDcard. The filename is automatically chosen to be the next free (unused) name of the form `NASxxx.BIN` where `xxx` is a 3-digit decimal number.

As the copy proceeds, a "." is printed for each successful (group of) reads and a "*" is printed for a failed (group of) reads. In case of a failed read, the copy should continue but the image will have a corresponding invalid region.

When the copy has completed, the same prompt is printed; you can insert a new floppy (which will be saved to a new filename).

To run this program you must be booted into disk PolyDos and have the nascom_sdcard hardware connected.

SCRAPE5

Like `SCRAPE` but assumes 35-track DSDD disks with 10 sectors per side, each of 512 bytes (so 35102*512=350Kbytes per disk). This is a CP/M disk format and I wrote this utility to transfer data from my CP/M floppies.

SETDRV

Display and change virtual drives on nascom_sdcard

```
$ SETDRV
```

Report the files mounted for each drive

```
$ SETDRV n filename
```

Unmount any SDcard file currently associated with drive (FID) n (0..3) and mount filename. filename must be a legal FAT "8.3" name. For example:

```
$ SETDRV 1 DRV0.BIN
```

In this example, drive 1 is now associated with the SDcard file `DRV0.BIN`

A bad drive number, missing filename, missing file extension or use of a file name that does not exist all result in an error being reported; the virtual drive mounts will be unchanged.

SDDIR

Perform a directory listing of the SDcard, with paging

```
$ SDDIR
```

Lists all of the files and directories in the (current directory of the) SDcard. After each screen of output, you are invited to "`Press [SPACE] to continue`". Pressing any other key will abort the listing.

CASDSK

Allows disk load/store for a program that was designed to use the W and R tape routines.

Intercepts the NAS-SYS W and R routines and redirects them to a single pre-defined disk file. Acts as a "terminate and stay resident" program and therefore must sit in free memory somewhere.

Example: Colossal cave adventure can "save" the game state using tape routines. Do this:

```
$ CASDSK CAVE.ME
Installed
$ COLOSSAL
```

Using SAVE and RESTORE within the program will still call W and R but now:

- W (write to tape) will actually write to the file CAVE.ME, deleting any pre-existing file of that name.
- R (read from tape) will result in the contents of CAVE.ME being loaded into memory at the address from which it was saved.

When installed and executed with no arguments, CASDSK is uninstalled; the normal R and W vectors are restored; the memory used by CASDSK can now be reused:

```
$ CASDSK
Uninstalled
```

When not currently installed and executed with no arguments, CASDSK just displays a message and terminates:

```
$ CASDSK
Not installed
```

CASDSK Implementation notes:

- The usual operation of PolyDos is to read and write data with a minimal granularity of 256 bytes. When saving, the same approach is taken: the write data is rounded up to the nearest 256 bytes. However, that may not be acceptable on reads, because it may overwrite data in memory. Therefore, on writes, the valid data size in the final sector (1-256 bytes) is stored in the low byte of the "execution address" entry of the data file (CAVE.ME in the example above). On reads, this size byte is used to transfer the file size.
- The algorithm can support any file size but the NAS-SYS calls that are intercepted are limited to a maximum size of 64Kbyte.
- This utility would work just as well on a real floppy-disk version of PolyDos.

SDOFF

Shut down the SDcard so that it (should be) quiescent on the PIO

```
$ SDOFF
```

The idea is to allow some other piece of hardware to use the PIO. Specifically, it was developed and tested with the Bits&PCs EPROM programmer.

Obviously, before running this program you need to get everything you need into memory (the EPROM programming software and the data to be programmed).

In order to restart the SDcard you need to reset it then reset the NASCOM and re-boot PolyDos. Even if you have (eg) uploaded an EPROM to RAM, it should be possible to restart and then save the EPROM image with no risk of corruption - certainly this seems to work reliably on my NASCOM 2.

Usage (NASdsk with NAS-DOS)

NAS-DOS requires 4Kbytes of code, stored in EPROM at $D000[3]. The NASdsk version of NAS-DOS has been derived from the disassembled source provided by Phill Harvey-smith (https://github.com/prime6809/NASDOS). NAS-DOS supports 4 virtual disk drive images each of 640Kbytes. Each virtual disk drive contains 2 virtual single-sided disks (each of 320Kbytes).

Boot the system by executing from the EPROM start address ($D000). The NASCOM screen should clear and display the startup banner:

```
-- NAS-DOS 1 -->D40238<-- NAS-SYS 3 --
```

NASdsk with NAS-DOS is in “beta test” - email me if you want access to the boot ROM source code. The `SETDRV`, `SDDIR` and `SDOFF` utilities (see page 10) should also work with NAS-DOS.

Usage (NASdsk with CP/M)

Not yet implemented! Let me know if you are interested (especially if you want to help code/test this)

Usage (NASdsk with DSKBOOT)

In order to use the NASdsk interface, you need some software on the NASCOM to talk to the PIO/"disk" interface. For example, PolyDos uses code in the PolyDos ROM. `DSKBOOT` is intended as a generic mechanism for this; it acts as a bootstrap loaded and makes it easy to use NASdsk for custom applications other than the PolyDos/NAS-DOS/CP/M ports.

The `DSKBOOT` source code is here: sdcard/host_programs/dskboot.asm

`DSKBOOT` is tiny (129 bytes). It is stored in the internal Flash filesystem so you do not need to copy it on your SDcard. Load and start it like this:

```
NAScas> RF DSKBOOT.GO
NAScas> .
R
(code loads at C80)
EC80 2 D000
-- NAS-DOS 1 -->D40238<-- NAS-SYS 3 --
```

3 NAS-DOS utilities are all hard-codes to use entry-points at the start of the $D000 region and therefore NAS-DOS cannot readily be relocated. The NASdsk version of NAS-DOS will run from RAM quite happily (the original NAS-DOS does not, because the `]V` command scribbles over memory at address $D000 upwards).

This tiny piece of code acts as a bootstrap loader. It mounts a disk image, loads the first sector of that disk image into RAM at $1000 and jumps into it.

The program `SDBOOT0.asm` is the code that is used for this boot sector. `SDBOOT0.DSK` is a concatenation of the `SDBOOT0` binary with various other binaries that are NASCOM ROM images.

`SDBOOT0` contains a data structure that describes the offsets, sizes and start addresses of these ROM images - so the concatenated `SDBOOT0.DSK` is like a very crude file-system.

All of this could be done by loading .CAS files but this is much faster..

When you start `DSKBOOT` by "`EC80`" it uses defaults in the data structure to load and execute various ROM images. You can override this because the `SDBOOT0` code looks at the arguments used to start `DSKBOOT`. So,

```
EC80 aaaa bbbb
```

aaaa is a bit-map of ROM images to load. bbbb is the address to go to after loading the ROM images (an address of 0 is decoded specially and does a NAS-SYS warm-start using SCAL MRET).

The default version of `SDBOOT0.DSK` contains these ROM images:

- bit 0: standard version of NAS-DOS, 4K @$D000
- bit 1: NASdsk version of NAS-DOS, 4K @$D000
- bit 2: standard version of PolyDos 2, 2K @$D000
- bit 3: NASdsk version of PolyDos2, 2K @$D800
- bit 4: ZEAP, 4K @ $D000
- bit 5: NASCOM ROM BASIC, 8K @$E000
- bit 6: PolyData PASCAL, ROM version, 12K @$D000
- bit 7: NAS-PEN, ROM version, 2K @ $B800

If you have RAM at $D000 you can modify your LKSW1 jumper to convert $D000 back to RAM (I put a tiny slider switch in) Now:

`EC80 8 D000`	-- loads the PolyDos ROM image and starts PolyDos
`EC80 28 D000`	-- loads the POLYDOS ROM image and the BASIC ROM image and starts PolyDos
`EC80 10 D000`	-- loads ZEAP and starts it
`EC80 2 D000`	-- loads NASDOS and starts it (you can get NAS-DOS disk images from nascomhomepage)

dskboot.asm and SDBOOT0.DSK are in host_programs/; SDBOOT0.asm and make_sdboot0.dsk show how SDBOOT0.DSK were created.

Usage (NASdsk utilies)

host_programs/ contains some development programs. They may be helpful for debug or as an aid to understanding how the system works. Each one is short enough that it can be assembled on a host system and typed in by hand in hex format (nostalgia). They all use a common set of subroutines, located at the start of the program. Having typed in one program, you can change to another simply by changing the non-common part at the end.

- sd_sub1.asm - common subroutines used by the other programs. Accessed by "including" this file.
- sd_loop.asm - test program that uses the loopback command to send values and check that they are received back correctly.
- sd_rd1.asm - test program for reading a file from SDcard into RAM
- sd_wr1.asm - test program for writing RAM to SDcard

Usage (NASdsk ROM utilities)

If you use polydos_util_rom.asm as the source-code for your boot ROM, there are 4 additional utilites that you can invoke from NAS-SYS. These utilities are executed through a jump-table at the end of the ROM. The execution addresses shown below assume a ROM assembled at address $D800:

- `E DFF4` -- CSUM
- `E DFF7` -- RDFILE
- `E DFFA` -- WRFILE
- `E DFFD` -- SCRAPE

CSUM

```
E DFF4 1000 800
```

Calculate and report a checksum of the $800 bytes starting at address $1000.

RDFILE

```
E DFF7 1000 34
```

Read file from SDcard into memory starting at address $1000. The transfer size is equal to the file size. The filename is `NAS034.BIN` - all but the number is hard-wired; the number comes from the last 3 digits of the argument (so `34`, `034` and `1034` would all result in the same filename).

WRFILE

```
E DFFA 1000 17FF AB
E DFFA 1000 17FF
```

Write data from memory address range 1000-17FF (hex, inclusive) to SDcard. In the first form, the filename is `NAS0AB.BIN` (see description above). In the second form, the filename is "auto-picked" -- the next unused name of the form `NASxxx.BIN` (where `xxx` are digits in the range 0..9) is chosen.

A note on auto-picked filenames: filenames explicitly specified by you can include hex digits, but the auto-pick algorithm (which runs on nascom_sdcard) uses decimal numbering. If you have created filenames `NAS001.BIN`, `NAS002.BIN`, `NAS003.BIN`, `NAS004.BIN`, `NAS005.BIN`, `NAS008.BIN`, `NAS009.BIN`, `NAS00A.BIN`, `NAS012.NAS` and then use auto-pick a few times, the auto-picked names will be `NAS000.BIN`, `NAS006.BIN`, `NAS007.BIN`, `NAS010.BIN`, `NAS011.BIN`, `NAS013.BIN`. Auto-pick will fail if all 1000 possible filenames are in use.

SCRAPE

```
E DFFD
```

Access the PolyDos floppy disk in drive 0 and copy its contents to a file on SDcard. The filename is "auto-picked" -- the next unused name of the form `NASxxx.BIN` (where `xxx` are digits in the range 0..9) is chosen.

Preparing an SDcard (NAScas)

NAScas only uses a single directory on the SDcard: either a directory named NASCOM (if it exists) otherwise the root directory.

NAScas can use PolyDos disk images or .CAS files or .TXT files. NAScas does not care about file extensions so you can use any extensions that you choose.

- If you have audio tapes you can recover data from them on a PC using the utility `wav2bin`
- Convert files between NAS/CAS/HEX/BIN format using the utility `nascon`
- Create and manipulate PolyDos disk images using the utility `polydos_vfs`

All of these utilities can be downloaded from: converters/. All of them are written in PERL. If you use Linux or Mac, you will already have PERL installed. If you use Windows I recommend Strawberry PERL (free download/install).

Pre-built PolyDos disk images can be downloaded from: PolyDos/lib

Preparing an SDcard (NASdsk)

NASdsk only uses a single directory on the SDcard: either a directory named NASCOM (if it exists) otherwise the root directory.

Data in the Arduino EEPROM controls the file names that are used for the disk images (see “NASdsk Profiles” on page 16).

In order to run PolyDos, copy all of the images from PolyDos/lib into the NASCOM directory of the SDcard and then copy `PD000.BIN` to `POLYDOS0.DSK`, and copy any other three images to `POLYDOS1.DSK`, `POLYDOS2.DSK` and `POLYDOS3.DSK`.

When starting PolyDos, nascom_sdcard tries to open 4 disk images, and the disk image corresponding to the boot drive (the drive you specified in response to the `Boot which drive?` prompt) must be bootable. The boot will fail if nascom_sdcard cannot find and open the 4 files that it expects.

The reason for using `PD000.BIN` is that it is bootable and contains all of the programs described in “PolyDos Utilities” on page 9. For example, you can use `SDDIR` and `SETDRV` from within PolyDos to inspect the SDcard and to mount different disk images.

In order to run NASDOS, copy 4 disk images into the NASCOM directory of the SDcard and name them `NASDOS0.DSK`, `NASDOS1.DSK`, `NASDOS2.DSK`, `NASDOS3.DSK`. Each image should be 655360 bytes in size. You can find NASDOS images on www.nascomhomepage.com.
When starting NASDOS, nascom_sdcard tries to open 4 disk images. The boot will fail if nascom_sdcard cannot find and open the 4 files that it expects.

In order to execute `DSKBOOT`, copy `SDBOOT0.DSK` from host_programs/ into the NASCOM directory of the SDcard then create files `SDBOOT1.DSK`, `SDBOOT2.DSK`, `SDBOOT3.DSK` – these can have any content (or even be zero-byte files).

When executing `DSKBOOT`, nascom_sdcard tries to open 4 disk images. The boot will fail if nascom_sdcard cannot find and open the 4 files that it expects.

The code that accesses the FAT filesystem on SDcard does not support long filenames, only 8.3 names. See “MSDOS FAT (8.3) Filenames” on page 6.

NASdsk Profiles

The EEPROM on the Arduino is programmed with a data structure called the “profile record”. The main function of the profile record is to define the file names that nascom_sdcard expects to find when starting up one of the software environments.

Each of the software environments has an associated number, called its profile, which is associated with a set of file names, as shown in the table below.

If the profile record is missing or corrupt, every profile will use the same set of file names (the “none” row in the table).

The idea of having different filenames for each profile is necessary, but the idea of storing this information in EEPROM was probably a waste of time.

Profile	Used for	File names
none	Anything	DSK0.BIN, DSK1.BIN, DSK2.BIN, DSK3.BIN
0	PolyDos	POLYDOS0.DSK, POLYDOS1.DSK, POLYDOS2.DSK, POLYDOS3.DSK
1	NASDOS	NASDOS0.DSK, NASDOS1.DSK, NASDOS2.DSK, NASDOS3.DSK
2	CP/M	CPM0.DSK, CPM1.DSK, CPM2.DSK, CPM3.DSK,
3	DSKBOOT	SDBOOT0.DSK, SDBOOT1.DSK, SDBOOT2.DSK, SDBOOT3.DSK

Theory of operation (NAScas)

NAScas provides storage for a NASCOM by interfacing to the (digital side of the) cassette interface. It works without the need for any expansion on the NASCOM or any on-board software, and work with all existing EPROM and tape-based software.

The serial interface can run at any true NASCOM baud rate (for authenticity) but the default setup is to generate a bit clock from the NAScas hardware in order to run the UART as fast as it and NAS-SYS can go. The current design runs at 2400bd on a NASCOM 2 and 600bd on a NASCOM 1.

Theory of operation (NASdsk)

The physical interface is a ribbon-cable connected to the PIO connector(s) of the NASCOM and drawing power from them.

The electrical interface uses 11 signals. 8 are used as a bi-directional data bus. CMD is a Command signal from the Host (NASCOM) to the Target (Arduino). H2T is a handshake from the Host to the Target, T2H is a handshake from the Target to the Host. +5V and Gnd connections complete the interface.

The communication protocol on the electical interface uses a handshake in each direction so that it is fully asynchronous (ie either end can run at any speed). Theoretically, the CMD signal is redundant, but it makes the protocol more rugged and easier to debug. The LED is illuminated if a data byte is received when a command byte is expected; it stays illuminated until the next command byte is received.

There are two parts to the software: software that runs on the Arduino, and software that runs on the NASCOM.

Software running on the Arduino is written in C using the Arduino IDE. It uses a pre-existing SD library to provide access to a FAT filesystem on the SD card. Use of the FAT filesystem means that files can be transferred to and from the card using modern mainstream computers.

Software running on the NASCOM is written in Z80 assembler. The Z80 program sends commands and data to the Arduino and reads status and data. The command-set (implemented on the Arduino) has been designed with the goal of keeping the Z80 software simple and small.

The communication protocol and the command-set are both described in detail in the “internals” documentation that you can find in doc/.

Construction

In the parts-list (below) the four columns at the right show which parts are needed for NAScas (N1 Ser/N2 Ser) or nascom_sdcard (N1 PIO/N2 PIO).

Reference	Description	N1 Ser	N1 PIO	N2 Ser	N2 PIO
	PCB	*	*	*	*
J5	6-pin microSD card adaptor	*	*	*	*
R1	4K7 resistor	*	*	*	*
R2	100K resistor	*		*	
R3	33R resistor	*		*	
D1	5mm red LED	*	*	*	*
D2	1N5817 Schotty Diode [REV B board only]	*	*	*	*
A1	Arduino Nano	*	*	*	*
C1, C2	0.1uF decoupling capacitor	*	*	*	*
C3, C4	10uF 16v electrolytic capacitor	*	*	*	*
J1	16-way 2x8 male pin IDC connector, polarised			*	
J2	6-pin 0.1” pitch header connector	*			
J3	26-way 2x13 male pin IDC connector, polarised (optional)				*
J4, J7	16 pin DIP 0.3” socket		*		
J6	26-way 2x13 male pin IDC connector, polarised				*

BEFORE assembly, check the revision of your PCB (marked top-side, to the left of the NASCOM logo). For a REV A board, refer to “REV A PCB ECO” below, and make the 4 PCB cuts described before inserting any components.

Fit the 3 resistors and 4 capacitors.

Fit the microSD card adaptor. Do not try to remove its 6-pin connector. Place the adaptor flat on the PCB so that the bottoms of the 6 pins sit in the PCB holes. They will not extend all the way through. Hold the board in place and run solder into the holes from the back of the PCB. This will make a solid electrical and mechanical connection. Trim the pins on the top of the adaptor so that they do not foul A1 and J3.

Fit the LED and (for REV B boards) diode D2.

Fit the connectors J1, J2, J3, J4, J6, J7 as required. Use polarised connectors where possible and follow the polarisation markings on the PCB. J3 is optional, even for the NASCOM2. Its purpose is to allow another device to be daisy-chained to the PIO (refer to the “SDOFF” command on page 11).

Fit the Arduino NANO. Be sure to get it the right way around: the USB connector is closest to the edge of the board, near hole H1. Refer to the photo on page 1. If you solder it directly to the board, I recommend not fitting it flush. If you raise it to leave 5-6mm of the pins showing; this will allow space to cut the leads (if you ever need to remove it) or to clip on a scope probe.

Finally, for REV A boards, fit the 4 wires of shame, described in “REV A PCB ECO” below.

REV A PCB ECO

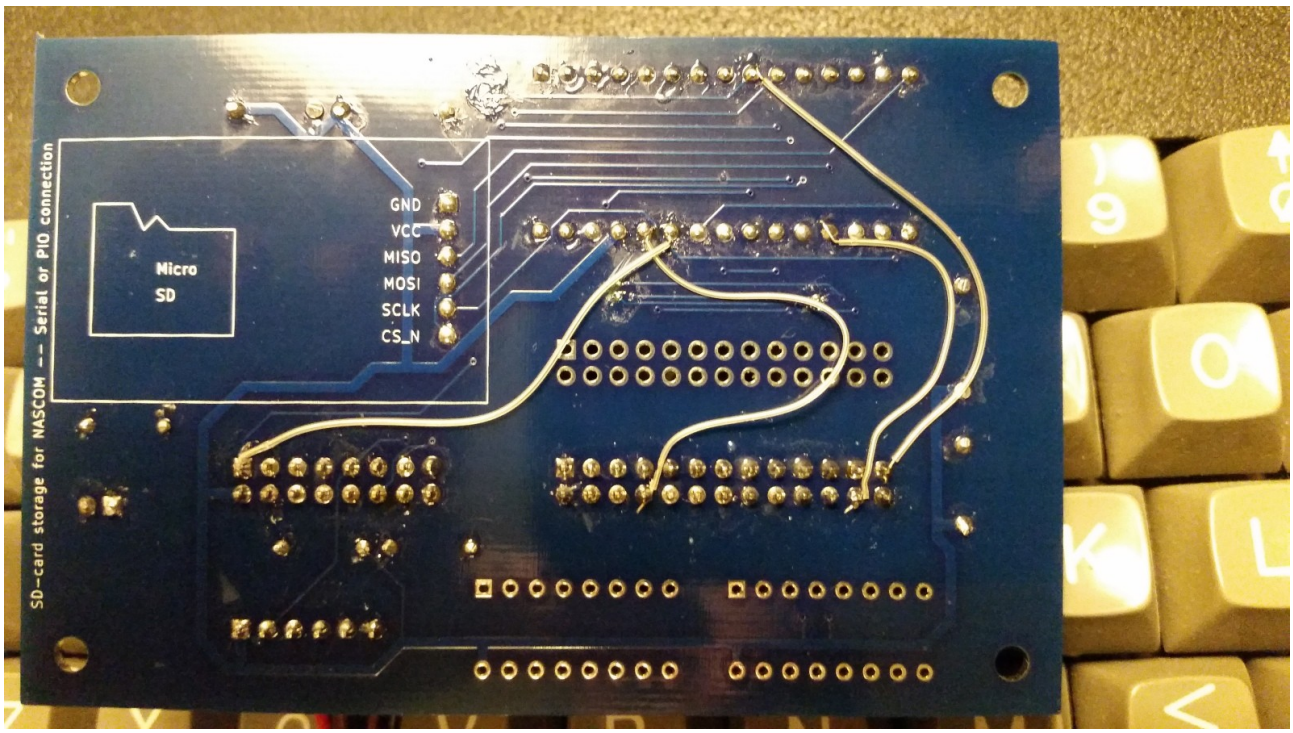
The “REV A” PCB requires 4 track cuts and 4 wire adds. Use a scalpel or similar to make the following cuts:

- Top-side A1/pin25 cut the track between the Arduino pad and the via.. A1 has 30 pins
- Top-side A1/pin26 cut the track between the Arduino pad and the via.. A1 has 30 pins
- Top-side A1/pin19 cut the track between the Arduino pad and the via.. A1 has 30 pins
- Bottom-side A1/pin9 cut the track between the Arduino pad and the via.. A1 has 30 pins

Use wire-wrap or other thin wire to make the following adds:

- Bottom-side add a wire from A1/pin25 to J1/pin1
- Bottom-side add a wire from A1/pin26 to J6/pin8
- Bottom-side add a wire from A1/pin19 to J6/pin24
- Bottom-side add a wire from A1/pin9 to J6/pin25

The pins of J1/J6 are *not numbered like an IC*; pin1 has pin2 to its right and pin 3 below it. See photo (below).



REV A / REV B Differences

There are 4 differences between the REV A and the REV B PCB:

- Some pin assingments to the Arduino were incorrect on REV A and had to be fixed with the ECO described above. No ECOs are required on REV B.
- A diode, D2, was added to REV B to isolate the NASCOM and Arduino 5V power rails. When the REV A board is connected both to a NASCOM and to USB, you must be careful to disconnect/power down the USB before powering down the NASCOM; otherwise, the USB will attempt to provide 5V power to the NASCOM which can burn out the protection circuit on the NANO. The diode D2 protects against this.
- A ground pin, J8, was added to REV B to allow easy attach of a scope ground lead.
- There was some minor tidy-up/rerouting of tracks.

The two boards are 100% software compatible.

Programming

This section covers programming of the Arduino board.

Install the Arduino IDE. Download from https://www.arduino.cc/

Install the SdFat library. Download from https://github.com/greiman/SdFat (usually, you will place the code in your ~/Arduino/libraries directory).

Download the github repository using one of these two methods:

- (Preferred method) git checkout https://github.com/nealcrook/nascom.git
- download https://github.com/nealcrook/nascom/archive/master.zip

The advantage of the preferred method is that you can go into this directory at any time and type “git update” to get the latest code from github.

Make the Arduino sketches available using one of these two methods:

- (Preferred method) In your Arduino project directory (~/Arduino) create a soft link sd_merged that points to the sdcard/sd_merged directory of the repository, and a second soft link setup that points to the sdcard/setup directory of the repository.
- In your Arduino project directory (~/Arduino) create a directory sd_merged and copy files from the sdcard/sd_merged directory of the repository, and a second directory setup and copy files from the sdcard/setup directory of the repository.

The advantage of the preferred method is that it avoids having a copy of the code, which might get out-of-date if you update the git repository from github.

From within the Arduino IDE, open the “setup” sketch. Select tools→board→Arduino Nano. (or Arduino Uno if you have reprogrammed the bootloader on your board: see notes below).

Use a USB cable to connect from your computer to the Arduino Nano. Click the Serial Monitor icon (magifying glass, top right in the GUI). Clickthe Upload (“→”) icon and check that the code builds and programs successfully. Once programmed, it will execute automatically, generating some messages in the Serial Monitor and finishing with the nascom_sdcard LED flashing. This completes the setup step.

```
LED test.. LED is off

LED test.. LED is on

LED test.. LED is off

Profile record checksum byte: 0x??

Profile record checksum: 0x??

Programming profile record..

Profile record checksum: 0x2F

Setup complete. Now program code from sd_merged

LED is flashing..
```

Next, from within the Arduino IDE, open the “sd_merged” sketch. If you’re building for NASCOM 1, edit the #defines at the start of the file. Click the Verify (“tick”) icon and check that it builds successfully. Ignore the following build warnings:

```
sketch/roms.h:1036:1: warning: initializer-string for array of chars is too long
[-fpermissive]

 };

 ^

sketch/roms.h:1036:1: warning: initializer-string for array of chars is too long
[-fpermissive]

sketch/roms.h:1036:1: warning: initializer-string for array of chars is too long
[-fpermissive]

sketch/roms.h:1036:1: warning: initializer-string for array of chars is too long
[-fpermissive]
```

Use a USB cable to connect from your compute to the Arduino Nano. Click the Upload (“→”) icon and check that the code builds and programs successfully. Once programmed, it will execute automatically, generating some messages in the Serial Monitor:

```
flags: 0x4009

prestore

using profile record

This is NAScas version 1.3 PROTO

Train..
```

Arduinos come with a “bootloader” pre-programmed into Flash. If you have the capability to reprogram the bootloader you can program the Uno bootloader into a Nano. The Uno bootloader is newer and smaller, so it leaves more Flash for program code.

When you are programming it, the Nano draws power from the USB. When it’s connected to the NASCOM, it draws power from the NASCOM. You can have it connected to both NASCOM and

USB at the same time – my experience is that you need to power up the NASCOM first before connecting the USB. With the NASCOM running and the USB connecting to a PC, start up the IDE and click the Serial Monitor (“Magnifying glass”) icon to open a serial terminal that reports debug/status information.

Installation (NASCOM 1)

NAScas needs to tap-in to the digital side of the UART interface. The tidiest way to achieve this is to add 4 wires to the back of the NASCOM 1, connecting the required signals to unused pins of the 16-way Serial Data Socket (SK2). The modified pinout of this connector is shown below (In/Out are shown with respect to the NASCOM):

```
                              1 --U-- 16 +5V
                  RS232  In   2 |   | 15
                              3 |   | 14 Out RS232 Out
                   KBD-  In   4 |   | 13
                   KBD+  In   5 |   | 12 Out PTR+
NEW Tape DRIVE (IC41/12) Out   6 |   | 11 Out PTR-
NEW      Uart In (LK3/In) In   7 |   | 10 Out Uart Out (IC29/25) NEW
               RS232 COM GND   8 ----- 9  In  Ext Cl P1 (LK4/P1) NEW
```

With these wires added, configure the NASCOM 1 links as follows:

- LK3: disconnect (so that the new pin 7 connection can drive serial data into the NASCOM)
- LK4: set to "Ext Cl" position (so that the new pin 9 connection can drive a serial clock into the NASCOM)
- LK2: fitted (single stop bit)

Use a DIL 16-pin header to connect from the NASCOM 1 SK2 to J2 on the SDcard interface PCB. Six connections are required:

Signal	Direction	NASCOM 1 connection
+5V	Power from NASCOM	SK2/16
RxD	In to NASCOM	SK2/7
Drive	Out from NASCOM	SK2/6
TxD	Out from NASCOM	SK2/10
Serial clock	In to NASCOM	SK2/9
Gnd	Common ground	SK2/8

To use NASdsk:

- Connect 2, 16-pin DIL/ribbon cables from nascom_sdcard to the NASCOM 1 “PIO” connectors (SKA, SKB). Be careful to orient the cable correctly at both ends.
- (Optional) Program and install appropriate boot EPROM(s). Refer to notes in the NASCOM 2 section, below.

Installation (NASCOM 2)

To use NAScas:

- Connect a 16-pin ribbon cable from nascom_sdcard to the NASCOM 2 “Serial” connector (PL2). Be careful to orient the cable correctly at both ends.
- Set the link switches. LSW1/5 to Up/On. LSW2 all switches to Up/On.

To use NASdsk:

- Connect a 26-way ribbon cable from nascom_sdcard to the NASCOM 2 “PIO” connector (PL4). Be careful to orient the cable correctly at both ends.
- PolyDos: Program and install a 2Kbyte EPROM. The source code is in host_programs/polydos_util_rom.asm and the same directory contains pre-build binaries to run at $D000 or $D800. This image fills the remaining ROM space with some useful utility programs.
- NASDOS: Program and install 4Kbytes of EPROM. Contact the author for the latest source code.

06Oct2019	Neal Crook, foofoobedoo@gmail.com	1st Edit.
13Oct2019	Neal Crook	Change NASCOM 1 connection for DRIVE.
17Nov2019	Neal Crook	Updates for NAScas 1.1: describe serboot changes, screen paging and 8.3 filename restrictions.
16Feb2020	Neal Crook	Add wiring table for NASCOM 1 installation.
19May2020	Neal Crook	Add inexplicably missing WS command.
23May2020	Neal Crook	Better details on how to create sdcard for PolyDos, corrections to RDFILE/WRFILE descriptions.
30Jul2020	Neal Crook	Move rev history to end. Revamp to reflect merge of serial and parallel interfaces and addition of console interface.