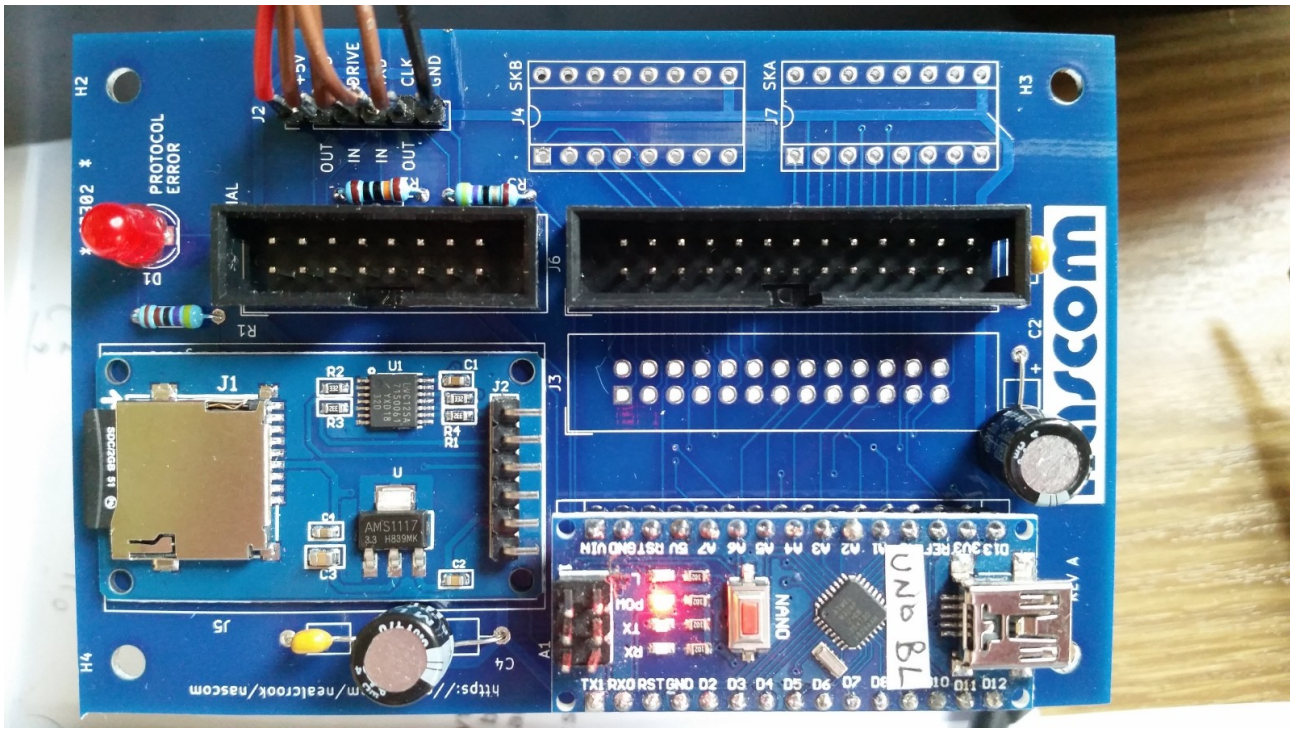
# NASCOM SDcard Interface




| | | |
|---|---|---|
| 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. |
| 22May2020 | Neal Crook | Better details on how to create sdcard for PolyDos |

## Introduction

This add-on for the NASCOM 1/NASCOM 2 provides file-based storage on a (micro) SDcard. It can be used in two different ways:

- 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 and runs PolyDos. This functionality is referred to as “nascom_sdcard”.

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.

It has been tested on a NASCOM 1 and on a NASCOM 2. All schematics, assembly drawings, software and application examples are freely available from: https://github.com/nealcrook/nascom

The hardware consists of an Arduino (UNO or NANO) board, a micro-SD Daughtercard, an LED and a few connectors and passives. I built the protoypes using an Arduino UNO and an Arduino prototyping PCB. The hardware aspects of this document refer to the “REV A” PCB which uses an Arduino NANO. The Arduino and micro-SD Daughtercard are cheap and easily available (eg from EBAY, Banggood or Aliexpress). The SDcard stores files in FAT format. The PCB is connected directly to the NASCOM using existing connectors and draws power from the NASCOM.

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.

# 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 (NAScas)

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

- it detects that the DRIVE signal has asserted
- it waits to see if data is received from the NASCOM (ie, a WRITE)
- if a timer expires and no data has been received, infer 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. 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 it will receive and process this command, starting the bootstrap program.

The SERBOOT source code is here:
https://github.com/nealcrook/nascom/sdcard/host_programs/serboot.asm

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 NAScas 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.

## 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 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 (in nascom/converters) for creating/maintaining PolyDos disk images and for converting between CAS and other formats.

# 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

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. 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 (nascom_sdcard)

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:
https://github.com/nealcrook/nascom/blob/master/PolyDos/doc/PolyDos2_doc.pdf

You can use any facility of PolyDos *except* the FORMAT or BACKUP commands.

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:
https://github.com/nealcrook/nascom/tree/master/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.

```
$ 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

Mounting a file that does not exist on the SDcard will create that file (of zero size). This is not a useful behaviour and could be considered a bug.

## SDDIR

Perform a directory listing of the SDcard, with paging

```
$ SDDIR
```

Lists all of the files and directories in the (root directory of the) SDcard. After each screen of output, you are invited to "Press [SPACE] to continue". There is no abort; you must page through the whole 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 andW 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 wad 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.

## Other nascom_sdcard utilities

The same github directory 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

## nascom_sdcard ROM utilities

If you used 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 1234
```

Read file from SDcard into memory starting at address $1000. The transfer size is equal to the file size. The filename is NAS234.BIN - all but the number is hard-wired, and the number comes from the three digits come from the last 3 digits of the argument (1234 in this example).

### WRFILE

```
E DFFA 1000 800 2345
E DFFA 1000 800
```

Write $800 bytes from memory to SDcard, starting at memory address $1000. In the first form, the filename is NAS345.BIN (see description above). In the second form, the filename is "auto-picked" -- the next free name of the form NASxxx.BIN (where xxx are digits in the range 0..9) is chosen.

### 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 free name of the form NASxxx.BIN (where xxx are digits in the range 0..9) is chosen.

## Preparing an SDcard (NAScas)

For NAScas, I recommend creating a directory named NASCOM and putting all the files in that 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: https://github.com/nealcrook/nascom/tree/master/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: https://github.com/nealcrook/nascom/tree/master/PolyDos/lib

# Preparing an SDcard (nascom_sdcard)

For nascom_sdcard put all the files in the root of the SDcard[1].

After reset, nascom_sdcard tries to open images named DSK0.BIN, DSK1.BIN, DSK2.BIN, DSK3.BIN as drives 0-3, respectively. The disk image corresponding to the boot drive (the drive you specified in response to the `Boot which drive? _` prompt) must be bootable.

I recommend copying all of the images from https://github.com/nealcrook/nascom/tree/master/PolyDos/lib onto the root directory of the SDcard and then copying PD000.BIN to DSK0.BIN, and any other images you chose to DSK1.BIN, DSK2.BIN and DSK3.BIN.

The reason for using PD000.BIN is that it is bootable and contains all of the programs described in “PolyDos Utilities” on page 7. For example, you can use SDDIR and SETDRV from within PolyDos to inspect the SDcard and to mount different disk images.

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.

# 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 (nascom_sdcard)

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.

1 I plan to change nascom_sdcard to use the NASCOM directory in the same way as NAScas but I have not yet made that change.

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 Arduino source code and in other diagrams/documents on my github.

Polydos uses a 2Kbyte boot EPROM, usually at $D000. The code in this EPROM provides low-level disk routines (like a BIOS) for the operating system. nascom_sdcard uses a modified EPROM image; no changes are needed to the disk-based parts of the code.

# 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 | * | * | * | * |
| 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, make the 4 PCB cuts described in the ECO section.

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 otherwise they will foul on A1 and J3.

Fit the LED.

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 elsewhere in this document).

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 fit the 4 wires of shame, described in the ECO section.

# ECO

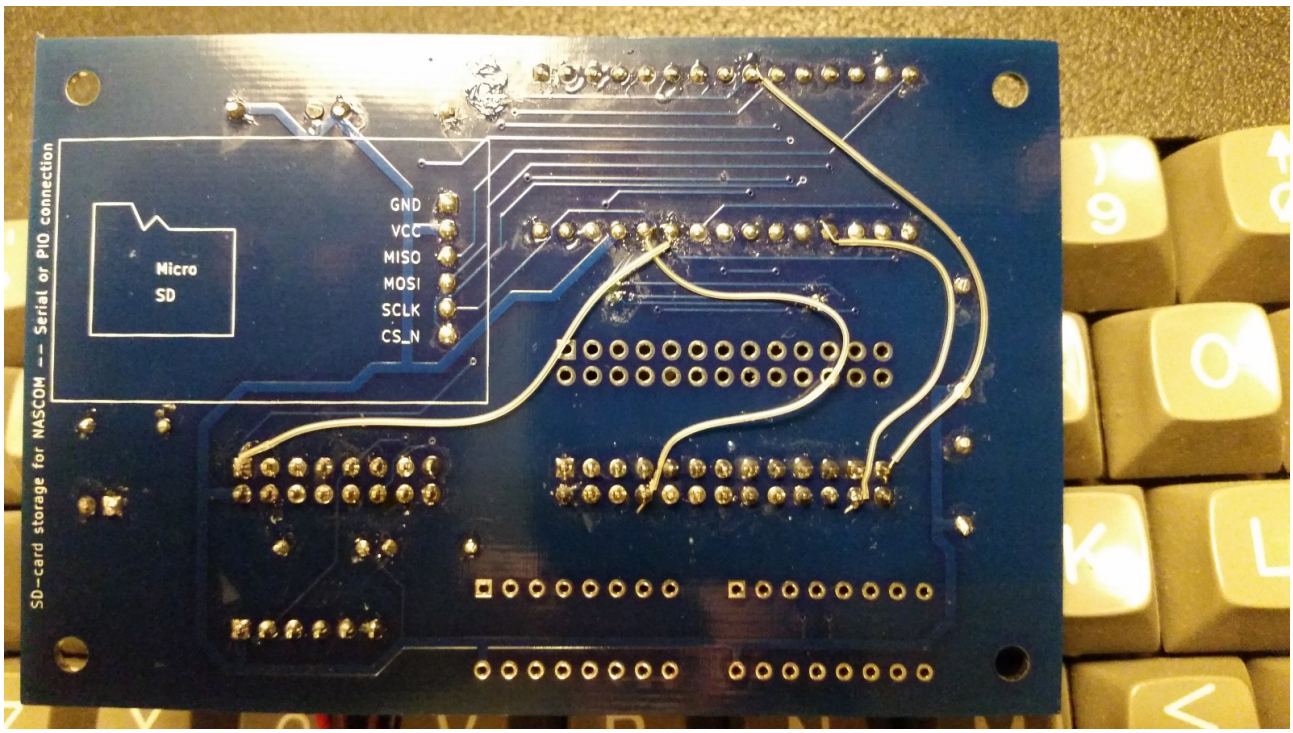
The “REV A” PCB requires 4 track cuts and 4 wire adds. Use a scalpel or similar to make the following cuts:

- 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).




# 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

In your Arduino project directory (Arduino/) create a NAScas/ directory and a nascom_sdcard/ directory and copy the files from https://github.com/nealcrook/nascom/tree/master/sdcard/NAScas and https://github.com/nealcrook/nascom/tree/master/sdcard/nascom_sdcard respectively.

From within the Arduino IDE, open the NAScas or the nascom_sdcard project. Select tools → board → Arduino Nano.

If you're building NAScas for NASCOM 1, edit the #defines at the start of the file. Click the Verify ("tick") icon and check that it builds successfully.

Use a USB cable to connect from your compute to the NANO. Click the Upload ("→") icon and check that the code builds and programs successfully.

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 NASCON, 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 nascom_sdcard:

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

## Installation (NASCOM 2)

To use NAScas:

- Connect a 16-pin ribbon cable from the SDcard interface PCB 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 swithes to Up/On.

To use nascom_sdcard:

- Connect a 26-way ribbon cable from the SDcard interface PCB to the NASCOM 2 “PIO” connector (PL4). Be careful to orient the cable correctly at both ends.
- Program and install a 2Kbyte EPROM. The source code (and pre-built binaries for location at $D000 or $D800) is available here: https://github.com/nealcrook/nascom/tree/master/sdcard/host_programs – if you rebuild it from source, use the source code polydos_util_rom.asm which fills the remaining ROM space with some useful utility programs.