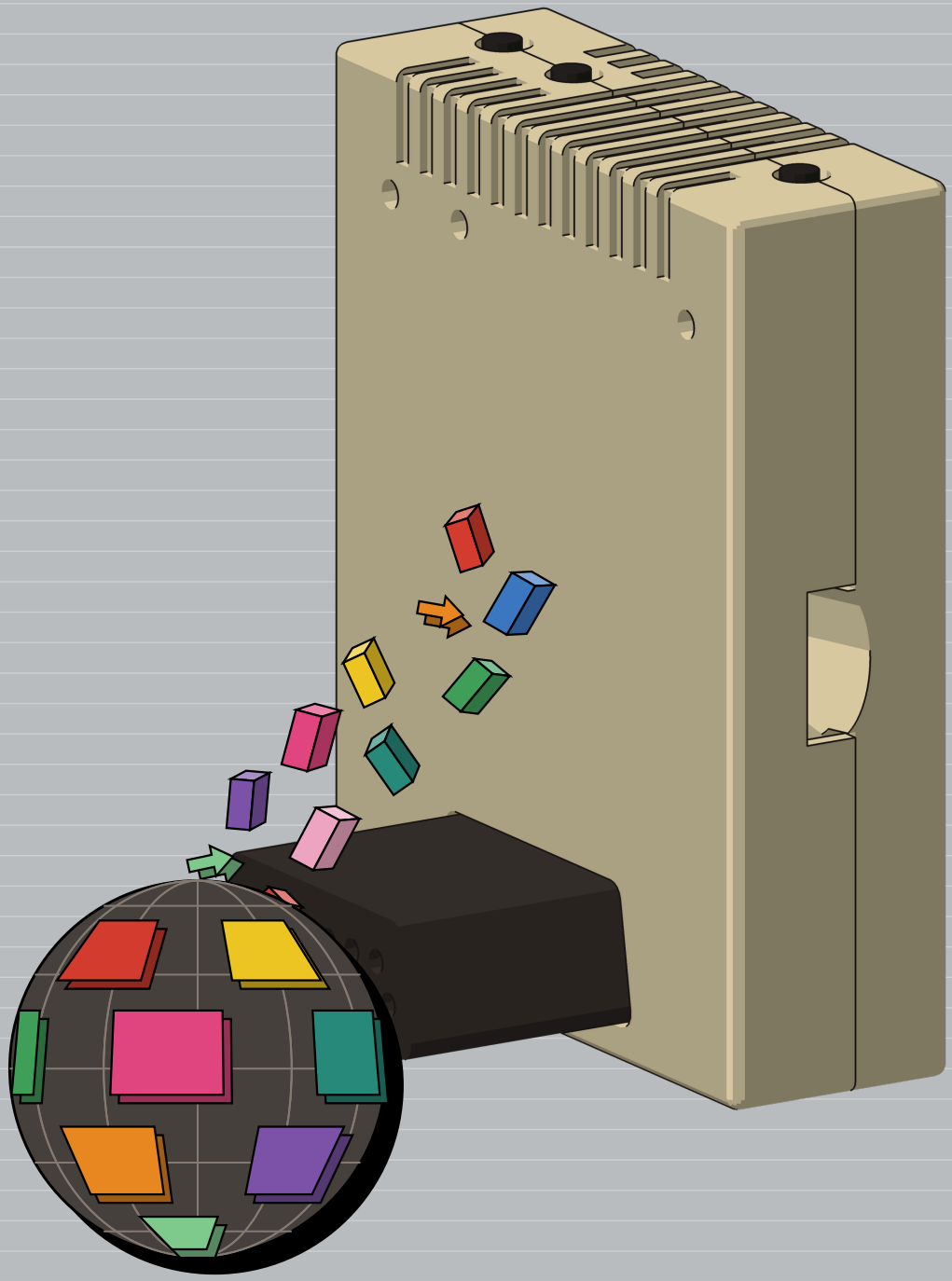
FUJINET™ NOS

AN INTRODUCTION TO THE NETWORK OPERATING SYSTEM



FUJINET

NETWORK OPERATING SYSTEM

CONTENTS

INTRODUCING NOS 2

BEGINNING WITH NOS 3

THE MENU 4

NETWORK, NOT DISK 5

CONNECTING TO A SERVER 6

THE EIGHT NETWORK DRIVES 7

THE PROTOCOLS 8

LOOKING AT THE DIRECTORY 10

WILD CARDS 12

FILESPECS AND URLS 13

SAVING AND LOADING A BASIC PROGRAM 14

LOADING PROGRAMS 15

COPYING FILES 16

DELETING AND RENAMING 17

TEXT FILES AND LINE ENDINGS 18

MOVING AROUND IN A FILE 19

BATCH FILES 20

AUTORUN: STARTING UP YOUR WAY 21

WHAT NOS DOESN'T DO 22

WHAT TO DO IF IT DOESN'T WORK 23

GETTING HELP 24

COMMAND REFERENCE 26

INSIDE NOS 29

APPENDIX: THE NOS SOURCE LISTING 31

INTRODUCING NOS

WHY YOU NEED NOS

The Network Operating System (NOS) is a program that allows your ATARI computer to work with your FujiNet™ and enables you to store and retrieve information on **servers** — other computers on your network and across the Internet. The information you save is still called a "file," and NOS (pronounced "noss") still lets you give a name to each file so you can call it up whenever you want it.

If you have used ATARI DOS with a disk drive, you already know most of what NOS does. NOS lists your files, deletes them, renames them, copies them, and loads and saves programs. The difference is where the files live. DOS keeps them on a diskette spinning a few inches away. NOS keeps them anywhere at all — on the computer in your den, or on a hobbyist server on the other side of the world.

WHAT YOU'LL LEARN FROM THIS BOOKLET

This is an introduction to NOS. After you read this booklet and follow the examples, you'll know your way around the NOS menu and the command line behind it: you'll connect a network drive to a server, look at its directory, and copy, delete, rename, load, save, and run the files you find there. You'll also learn how to make NOS set up all your favorite connections automatically, every time you turn the computer on.

A complete reference for every NOS command begins on the reference pages near the back of this booklet; a tour of NOS's insides — its memory map and its overlay machinery — sits just ahead of the full source listing, printed in the back for advanced users. NOS also carries its own online HELP library, described in the section "Getting Help."

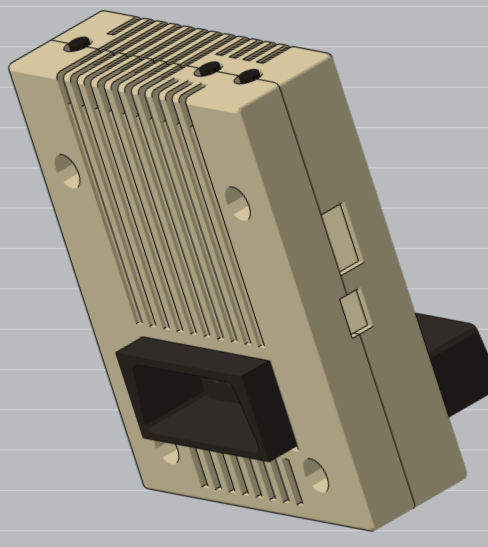
One thing you **won't** find here is anything about floppy diskettes. NOS doesn't use them. There is nothing to format, nothing to insert, and nothing to fill up.

GOT EVERYTHING?

Before you begin, check that you have:

1. An ATARI home computer, set up and working.
2. A FujiNet, connected to your wireless network. (The FujiNet owner's guide covers this.)
3. The NOS disk image, `NOS.ATR`. It is always available at `apps.irata.online` in the `Atari_8-bit/DOS/` folder.

NOS is young software, still growing. It is best suited to workflows where your ATARI and a modern computer work together — and it is developed in the open. The source code lives in the `fujinet-nhandler` repository on GitHub, under `nos/`.



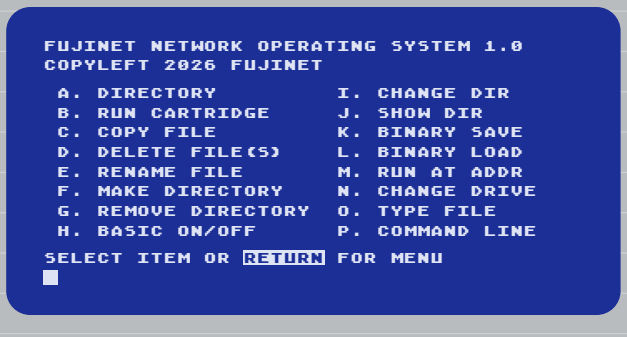
BEGINNING WITH NOS

HOW PROGRAMS ARE STORED ON THE NETWORK

Programs are stored in an area of the computer called **the memory,** and the memory forgets everything when you turn the power off. A disk drive fixes that by recording your work on a diskette. Your FujiNet fixes it by sending your work through the air to a server — any computer, anywhere, that agrees to hold files for you.

When you save a file under NOS, a stream of information flows out of the computer's memory, through the FujiNet, across your wireless network, and comes to rest on the server you chose. Loading is the reverse: the server streams the file back, and it settles into the computer's memory. If there was a program in memory already, the new one is written over it.

The server holding your files is called a **mount,** and the connection NOS makes to it is called a **network drive.** You have eight of them, named **N1:** through **N8:** — more about them in a few pages.



LOADING NOS

NOS boots from a disk image, the same way DOS boots from a diskette — your FujiNet simply pretends to be the drive.

1. Boot your FujiNet CONFIG program, and navigate to the host **apps.irata.online**.
2. Find **NOS.ATR** in the folder **Atari_8-bit/DOS/** and mount it in disk slot 1, read-only is fine.
3. Boot the disk. In a moment you'll see the NOS menu — sixteen choices, waiting on one keystroke.

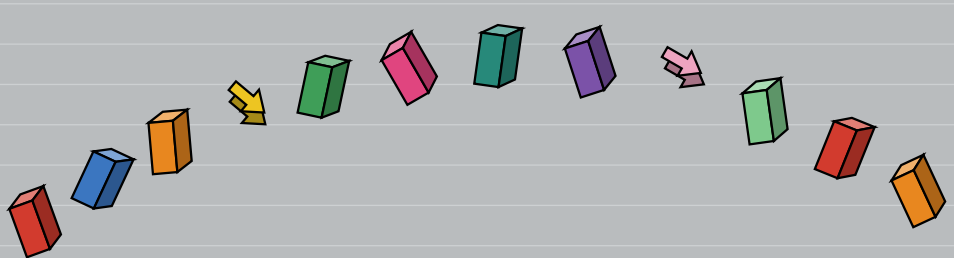
THE MENU

What greets you is the NOS menu, drawn from the school of DOS 2.0: sixteen choices, one letter each. Press a letter, answer the question it asks, and NOS does the rest — nothing to remember, nothing to spell. The next section walks through every letter.

The last item is a door. **P. COMMAND LINE** opens the NOS prompt, where every menu item is a command you can type with more say in the matter — and where a dozen commands live that never made the menu. This booklet teaches both, letter and command together; type **MENU** at the prompt whenever you want the menu back.

LEAVE THE NOS DISK IN THE DRIVE

Like DOS, NOS keeps only part of itself in memory. The everyday machinery stays resident; the less-used commands — and the menu itself — wait out on the NOS disk image, loaded in the moment you call them. So leave **NOS.ATR** mounted in disk slot 1 while you work. Where DOS swapped a whole menu program over your work (and invented MEM.SAV to apologize for it), the NOS menu is ten small sectors borrowing one small patch of free memory — and the command line borrows nothing at all. The chapter "Inside NOS," near the back, maps every byte.



THE MENU

Sixteen letters, and every one of them is a NOS command wearing a nametag. Choose a letter and the menu asks one plain question — some of them word for word the questions DOS 2.0 asked. Answer it, and the menu quietly assembles the real command, runs it, and waits beneath the result:

```
COMPUTER: SELECT ITEM OR RETURN FOR
          MENU
YOU TYPE: A RETURN
COMPUTER: DIRECTORY-SEARCH SPEC?
YOU TYPE: *.XEX RETURN
COMPUTER: GAME.XEX          25K
        : DEMO.XEX          12K
```

Answer a question with a bare RETURN and the item runs plain — item A with no search spec lists everything. When the result ends, the SELECT ITEM prompt returns beneath it, your listing still on the screen; press RETURN alone and the whole menu redraws.

WHAT EACH LETTER ASKS

ITEM	IT ASKS	RUNS
A. Directory	DIRECTORY-SEARCH SPEC?	**DIR**
B. Run Cartridge	— runs at once —	**CAR**
C. Copy File	COPY-FROM,TO?	**NCOPY**
D. Delete File(s)	DELETE FILE SPEC?	**DEL**
E. Rename File	RENAME-OLD,NEW?	**RENAME**
F. Make Directory	MAKE DIRECTORY?	**MKDIR**
G. Remove Directory	REMOVE DIRECTORY?	**RMDIR**
H. BASIC On/Off	BASIC ON OR OFF?	**BASIC**
I. Change Dir	CHANGE TO DIRECTORY?	**NCD**
J. Show Dir	— runs at once —	**NPWD**
K. Binary Save	SAVE-NAME,START,END?	**SAVE**
L. Binary Load	BINARY LOAD FILE?	**LOAD**
M. Run At Addr	RUN AT ADDRESS (HEX)?	**RUN**
N. Change Drive	CHANGE TO DRIVE (1-8)?	**Nn:**
O. Type File	TYPE FILE?	**TYPE**
P. Command Line	— opens the prompt —	

THE DOS MENU, REBORN

If you grew up on ATARI DOS, look closely: the letters you wore into muscle memory still do what they always did. A lists the directory, C copies a file, D deletes, E renames, K and L save and load binaries, M runs at an address, B runs the cartridge.

Only the chores that came with diskettes gave up their letters. FORMAT DISK and DUPLICATE DISK are gone — nothing to format, nothing to copy sector by sector — and LOCK, UNLOCK, WRITE DOS FILES, and CREATE MEM.SAV went with them. Their letters were handed to network work: directories to make and remove, a drive to change, a file viewer, and the door to the command line.

And one old letter grew truer. **D. DELETE FILE(S)** finally earns its plural: give it a wild card like ***.BAK** and it works through the whole crowd, asking about each file by name. (The chapter "Wild Cards" tells that story.)

WHERE THE MENU LIVES

The menu is not part of resident NOS. Each time it draws, NOS reads it fresh from the disk image — ten small sectors — into a patch of free memory at $2600, and your chosen command runs from **resident** code, so even a command that overwrites the menu can't get lost on the way home. The price is honest and small: a program tall enough to reach $2600 — about 2.8K past NOS's floor — gets a corner stepped on when the menu draws. The command line loads nothing up there at all, which is one more reason **P** is the power user's first keystroke. "Inside NOS," near the back, has the whole map.

NETWORK, NOT DISK

A DISKETTE THAT CAN BE ANYWHERE

Every disk operating system before this one was built around a piece of spinning plastic. NOS is built around a connection. Where DOS thinks "drive 1, sector 42," NOS thinks "that folder on that server." Once a network drive is connected — **mounted,** as NOS says — everything you know from DOS carries over: the directory, the filenames, copying, deleting, renaming, loading, saving.

A mounted server behaves like a diskette with a few upgrades. It can be as big as the server's owner allows. Several people can reach it at once. And it never goes soft in a hot car.

WHERE DID D: GO?

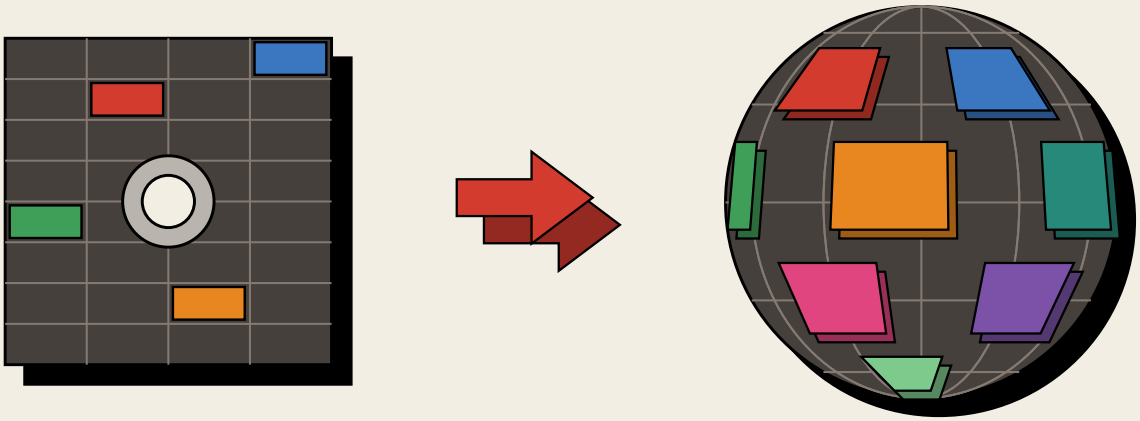
ATARI software talks to diskettes through a device named `D:`. NOS contains no disk software at all — no File Management System, no sectors, no formatting — but it still answers calls to `D:`. When a program asks `D:` for a file, NOS quietly hands the request to the network device, `N:`. The program never knows the difference.

This is how a BASIC program written in 1982 can `SAVE "D:MYFILE"` and have MYFILE come to rest on a server in another hemisphere. The program thinks it's talking to a 1050. It's talking to the world.

WHAT THAT MEANS DAY TO DAY

- Drive numbers name **connections,** not hardware. `N1:` through `N8:` can each point somewhere different.
- There are no BASIC pokes, no handlers to load, no cartridges. NOS installs its `N:` handler at boot, and maps `D:` to it.
- Physical and emulated diskettes are **not** reachable from NOS. If you need real disk images, boot a disk-based DOS instead — see "What NOS Doesn't Do."

The next few pages teach the one genuinely new skill NOS asks of you: connecting a network drive to a server.



The diskette you know, and the diskette you're getting: a mounted server holds files the same way, listed the same way, named the same way — it just isn't in the room.

CONNECTING TO A SERVER

Before a network drive can do anything, it must be connected to a server. The command that does it is **NCD** — **network change directory** — item **I. CHANGE DIR** on the menu, and if the two-letter version suits you better at the prompt, **CD** does the same thing.

You hand NCD a **URL:** the name of a protocol, the name of a host, and a path. (Protocols get their own section, a few pages on. For now, TNFS is the protocol FujiNet folks use most.)

YOU TYPE: `NCD N1:TNFS://192.168.1.20/` RETURN

COMPUTER: `N1:`

No news is good news: the prompt returns, and drive 1 is mounted. If NOS can't reach the server, you'll get an error number instead. To see where a drive points, ask **NPWD** (or **PWD** — item **J** on the menu):

YOU TYPE: `NPWD` RETURN

COMPUTER: `N1:TNFS://192.168.1.20/`

MOVING AROUND

Once a drive is mounted, NCD moves through the server's folders the way you'd move through directories on any big computer. Relative paths work, and so does `..` for the parent directory:

YOU TYPE: `NCD GAMES/ACTION` RETURN

YOU TYPE: `NPWD` RETURN

COMPUTER: `N1:TNFS://192.168.1.20/GAMES/ACTION/`

YOU TYPE: `NCD ../PUZZLE` RETURN

YOU TYPE: `NPWD` RETURN

COMPUTER: `N1:TNFS://192.168.1.20/GAMES/PUZZLE/`

A trailing `/` is added for you if you leave it off.

NAMES WITH SPACES

If a path has spaces in it, wrap the whole thing — device, URL, and all — in double quotes:

YOU TYPE: `NCD "N2:FTP://ftp.pigwa.net/stuff/holmes cd/"`

LETTING GO

To disconnect a drive, give NCD a bare drive name and nothing else:

YOU TYPE: `NCD N2:` RETURN

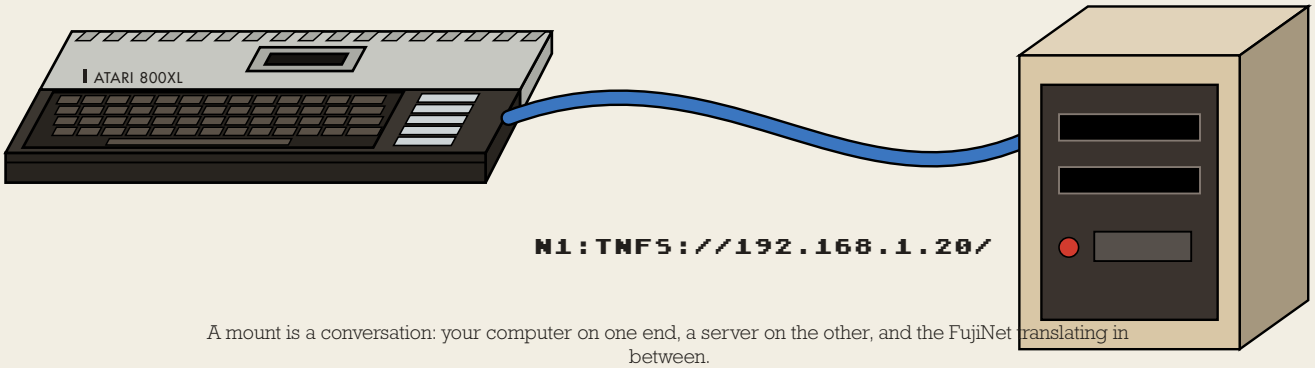
WHICH DRIVE GETS MOUNTED?

If the URL starts with a drive name like `N2:`, that drive is mounted. If it starts with the bare protocol, the **current** drive — the one in your prompt — is mounted. Both of these connect drive 2:

YOU TYPE: `NCD N2:TNFS://192.168.1.20/`

YOU TYPE: `N2: then NCD TNFS://192.168.1.20/`

One caution before you wander: NCD does **not** check that a new path really exists on the server. If you mistype a folder name, the mistake shows up later, when DIR or LOAD comes back empty-handed. When in doubt, DIR right after you arrive.



A mount is a conversation: your computer on one end, a server on the other, and the FujiNet translating in between.

THE EIGHT NETWORK DRIVES

Your FujiNet carries **eight** network devices, **N1:** through **N8:**, and every one of them can be mounted somewhere different at the same time. One drive on your workroom TNFS server, one on an FTP archive in Poland, one on the FujiNet's own SD card, one reading a page from the web — all at once, all switchable with two keystrokes.

To make another drive the current one, press **N** at the menu and answer with a digit — or, at the prompt, type the drive's name, alone:

YOU TYPE: **N3:** RETURN

COMPUTER: **N3:**

The prompt follows you. Commands typed without a drive name — **DIR**, **NCD PATH**, **LOAD GAME.XEX** — now mean drive 3. Switching is allowed even if the drive has no mount yet; NOS doesn't check until you try to use it.

THE MOUNTS LIVE IN THE FUJINET

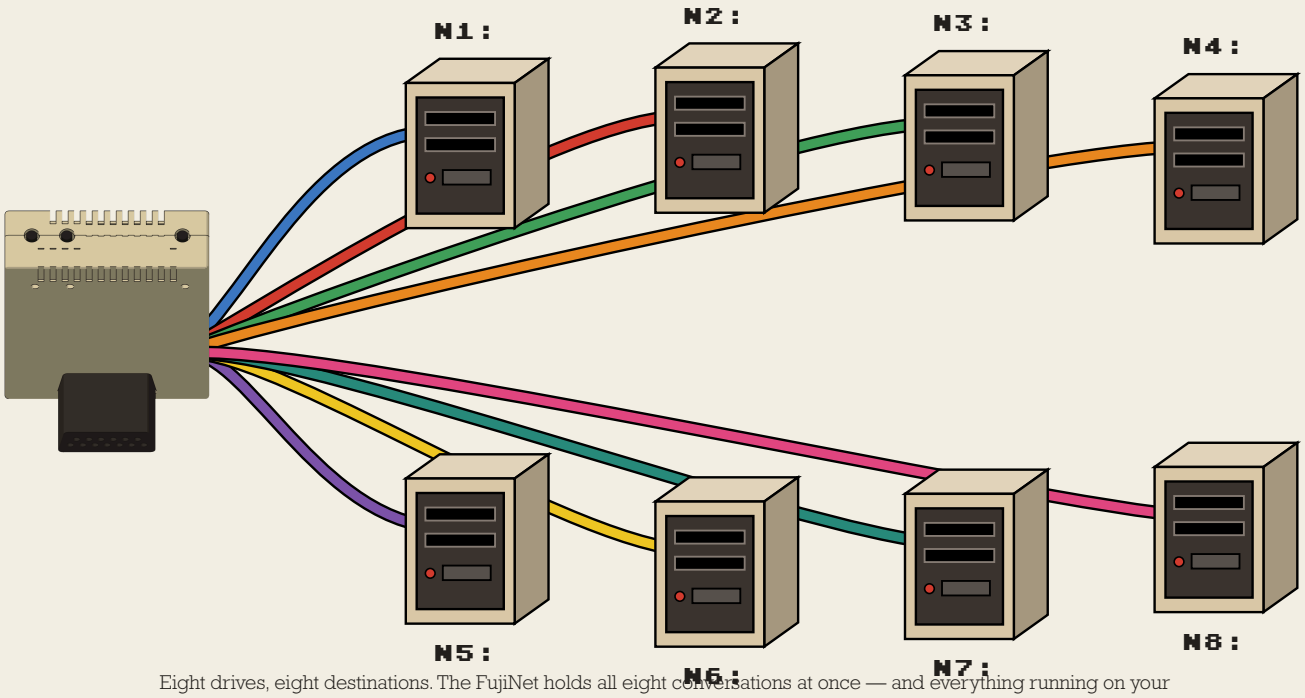
Here is the single most important idea in this booklet. The mount table — which drive points where — is kept **inside the FujiNet,** not inside your computer. NOS is only one voice giving it instructions.

That has a happy consequence: mounts survive. Load a program, quit back to NOS, and your drives are still connected right where you left them.

And it has a consequence to respect: the drives are **shared.** A BASIC program that opens **N1:** sees exactly the same **N1:** the NOS prompt sees. If you — or a batch file, or the program itself — **NCD** drive 1 somewhere else, it moves for **everyone,** including any program that was counting on it.

ADVICE FOR PROGRAMMERS

- Pick your drives deliberately. If your program keeps its data on **N1:**, do your NOS housekeeping on another drive.
- Be careful with **NCD** (or **CD**) inside programs and batch files — you are moving a shared drive, not a private one.
- Leave **N4:** to NOS when you can. NOS itself borrows drive 4 as its service line: HELP fetches its articles over it, and NCOPY builds its network destinations there. A mount you park on **N4:** can confuse both.
- Keep everyday file traffic on drives 1 through 4. All eight drives answer the drive commands (NCD, NPWD, DIR, DEL, RENAME, MKDIR, RMDIR, NTRANS, LOAD), but the resident **N:** handler provides stream buffers for units 1–4, and that is where TYPE, NCOPY, SAVE, SUBMIT, and your own OPEN statements should live.



Eight drives, eight destinations. The FujiNet holds all eight conversations at once — and everything running on your ATARI shares them.

THE PROTOCOLS

A **protocol** is the language a server speaks. You choose one in the first word of every URL, before the `://` — and that's the only place you ever deal with it. Once a drive is mounted, every protocol looks like the same thing: files.

The FujiNet firmware speaks all of the protocols on this page. They differ in what they allow. Some let you do everything DOS ever did — list, read, write, delete, rename, make and remove directories. Others are read-mostly, and a couple are special guests. The table below is the map.

TNFS — THE HOME TEAM

TNFS was invented for machines like yours. Servers are free and tiny — run `tnfsd` on any PC, Mac, or Linux box — and most public FujiNet software libraries speak it, including `apps.irata.online` and `tnfs.fujinet.online`. Everything works: reading, writing, the whole DOS toolbox.

YOU TYPE: `NCD N1:TNFS://192.168.1.20/`

If a URL gives no protocol port, the standard one is used (16384 for TNFS), which is nearly always right.

SD — THE SERVER IN YOUR HAND

The protocol named `SD` is a server that never leaves home: the microSD card plugged into your FujiNet. No host name needed — and no network, either. Everything works, just like TNFS.

YOU TYPE: `NCD N1:SD://fujinet/`

FTP — THE DEEP ARCHIVES

Decades of ATARI software sit on FTP sites. NOS signs you in as an **anonymous** guest — fine for public archives, which are usually read-only to guests. Directory listings, downloads, and (where the server permits) uploads all work.

YOU TYPE: `NCD N2:FTP://ftp.pigwa.net/atari/`

HTTP AND HTTPS — THE WEB ITSELF

Any web URL can be opened and read — TYPE a page straight off the Internet. The full DOS toolbox (directories, writing, renaming, deleting) works when the far end is a **WebDAV** server; an ordinary web server is read-only.

YOU TYPE: `TYPE N3:HTTPS://fujinet.online/`

SMB — THE HOUSE NETWORK

SMB is the file sharing built into Windows (and served by Samba and most NAS boxes). Everything works. If the share wants a name and password, supply them with the USER and PASS commands **before** mounting:

YOU TYPE: `USER MOLLY` RETURN

YOU TYPE: `PASS SECRET` RETURN

YOU TYPE: `NCD N1:SMB://DEN-PC/ATARI/`

NFS — THE UNIX COUSIN

The traditional file sharing of Unix machines. Everything works.

YOU TYPE: `NCD N1:NFS://192.168.1.9/export/atari/`

GDRIVE — A CLOUD GUEST

Google Drive, by way of a relay at fujinet.online. Authorize your Google account once in the FujiNet's web configuration page, and your Drive becomes a mountable volume — list, read, write, delete, make directories. (Renaming isn't provided.)

YOU TYPE: `NCD N1:GDRIVE://drive/atari/`

WHAT WORKS WHERE

PROTOCOL	DIR	READ	WRITE	DEL / REN / MKDIR	NOTE / POINT
TNFS	■	■	■	■	■
SD	■	■	■	■	■
FTP	■	■	(1)	(1)	—
HTTP(S)	(2)	■	(2)	(2)	(4)
SMB	■	■	■	■	■
NFS	■	■	■	■	■
GDRIVE	■	■	■	(3)	—

(1) where the server permits its guests. (2) needs a WebDAV server; plain web sites are read-only. (3) everything except RENAME. (4) reading only — see "Moving Around In a File."

BEYOND FILES: THE STREAM PROTOCOLS

The `N:` device speaks a few more languages — `TCP:`, `UDP:`, `TELNET:`, `SSH:` — that carry live **streams** instead of files. There is no directory to list, so DIR and friends shrug; but a program (or a bold TYPE command) can hold a conversation through them. They belong to the FujiNet programmer's guide rather than to this booklet.

MIND YOUR CASE

One habit to unlearn from DOS: most servers care about capitalization. On a TNFS or SMB server, `Jumpman.xex` and `JUMPMAN.XEX` are different files. When a command comes back empty-handed, check the case of what you typed against DIR.

LOOKING AT THE DIRECTORY

The Directory contains a list of all the files at the current spot on a mounted network drive. To see it, press **A** at the menu and answer the search-spec question with a bare RETURN — or, at the prompt, type **DIR**:

YOU TYPE: **DIR** RETURN

Each line shows a name and, at the right, a size — plain bytes for small files, **K** for kilobytes, **M** for megabytes. Folders you can NCD into are marked with a trailing **/**. Servers can hold **far** more than a diskette's 707 sectors, so long listings scroll: hold SPACE to pause the parade, and press ESC to stop it.

A directory somewhere else is no harder. Name the drive, or the path, or both:

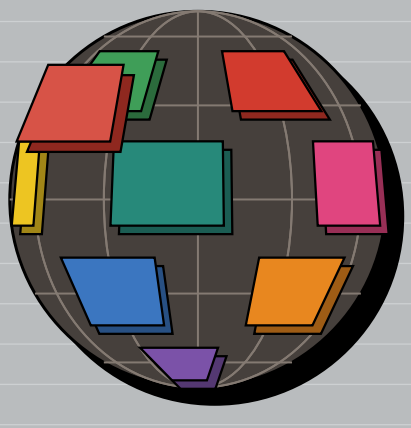
YOU TYPE: **DIR N2:** RETURN

YOU TYPE: **DIR GAMES/** RETURN

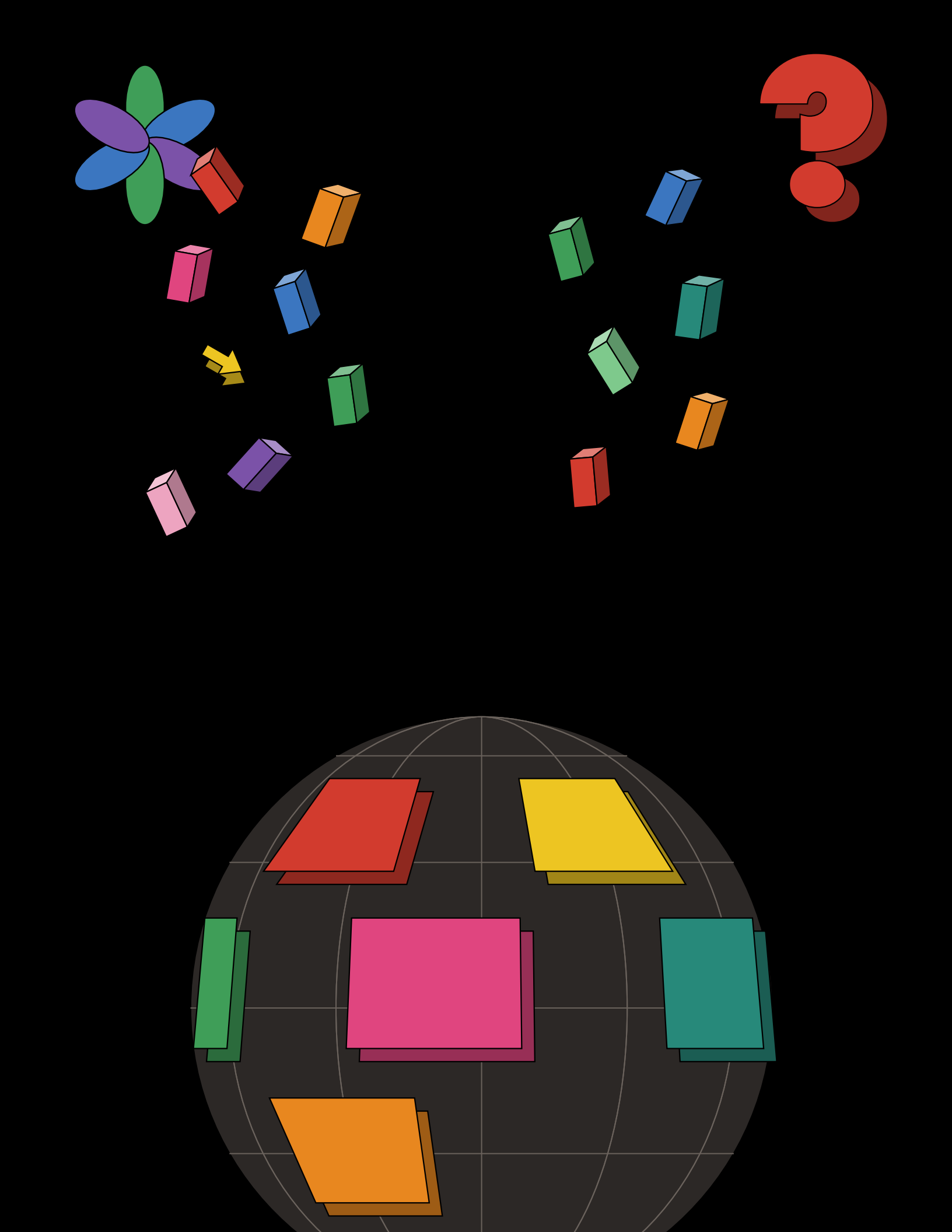
YOU TYPE: **DIR N2:GAMES/*.XEX** RETURN

That last one uses a **wild card** — the subject of the next section.

```
N1:DIR
JUMPMAN.XEX                25K
STAR.RAIDERS.XEX           33K
HELLO.BAS                 512
NOTES.TXT                1201
LETTER.TXT               4400
GAMES/
UTILS/
N1:
```



The directory is the map of the volume: every file, its size, and the folders inside.



WILD CARDS

A wild card in a filename works just like a joker in a pack of cards — it stands in for characters you don't feel like spelling out. Two are allowed: an asterisk (*****) stands for any run of characters, and a question mark (**?**) stands for exactly one.

Suppose your work directory holds a season's worth of BASIC programs mixed in with text files. To see just the BASIC:

```
YOU TYPE:  DIR *.BAS  RETURN
COMPUTER:  FILE1.BAS      884
       :   FILE2.BAS      901
       :   GAME.BAS       15K
```

And to pick out numbered files one digit apart:

```
YOU TYPE:  DIR NAME?.DAT
           RETURN
COMPUTER:  NAME1.DAT      128
       :   NAME2.DAT      128
       :   NAME3.DAT      128
```

Two small prints. First, folders are always listed, whatever pattern you give — the pattern sifts files only. Second, remember that servers usually mind capitalization: ***.xex** and ***.XEX** can be different crowds.

WHERE WILD CARDS WORK

Wild cards work in **DIR** — and, new in NOS 1.0, in **DELETE** and **COPY** too. Deleting stays careful: every matched file is offered back by name, and only a Y sends it away. Copying announces each file as it goes. The chapters "Copying Files" and "Deleting and Renaming" show both at work. **RENAME** is the lone holdout: one plain filename at a time.

One grain of fine print: NOS gathers the matched names into a 512-byte basket — room for a few dozen names per command. If a pattern catches more files than the basket holds, simply run the command again; it refills with the files that remain.

FILESPECS AND URLS

Just as you call a person by name, so must you call a file by its right name. A complete NOS filespec has more parts than a DOS one, because it can name any file on Earth — but you'll almost never spell out the whole thing. Once a drive is mounted, the drive remembers the protocol, the host, and the path, and a filespec shrinks back to the friendly old shape:

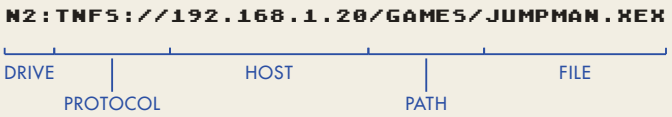
YOU TYPE: `LOAD JUMPMAN.XEX`
YOU TYPE: `LOAD N2:GAMES/JUMPMAN.XEX`

THE RULES

- Names may be long, and may use upper case, lower case, digits, and punctuation the server allows.
- Most servers treat capital and small letters as **different.** `Letter.txt` is not `LETTER.TXT`.
- A name with spaces must ride inside double quotes, device and all: `DEL "N2:My File.TXT"`
- Extenders still mean what they meant: `.BAS` for BASIC, `.XEX` and `.COM` for machine programs, `.TXT` for text.
- Paths use `/`, and `..` walks up one folder.

ANATOMY OF THE FULL SPEC

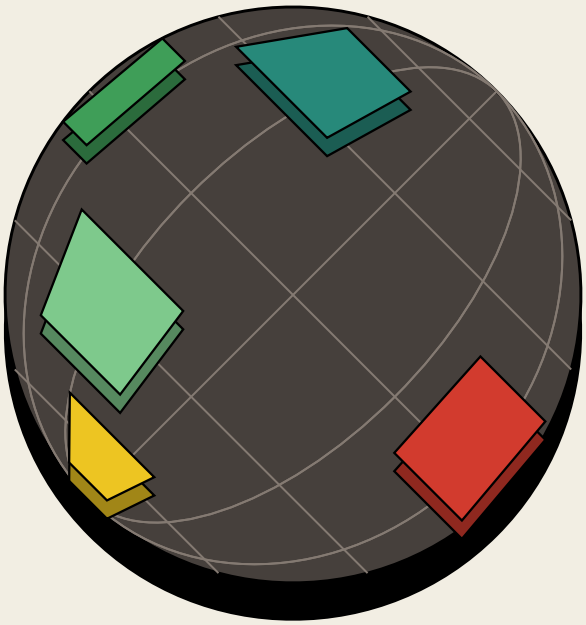
When you do need the whole thing — in NCD, or in a filespec aimed at an unmounted spot — the parts snap together like this:



The drive says **which of the eight connections.** The protocol says **what language.** The host says **which server** — a name or an address, with an optional `:port` after it if the server listens somewhere unusual. The path and file say the rest.

WHEN D: APPEARS

Programs may still say `D:` or `D1:` in their filespecs. NOS forwards the call to the matching network drive — `D1:` means `N1:`, `D2:` means `N2:`, and so on. Nothing to set up; it simply works.



SAVING AND LOADING A BASIC PROGRAM

It's quite easy to save a program you have written onto a server, and then load it back — even for a program that has never heard of a network. Let's prove it with the oldest two-liner in the book. You'll need a mounted drive you're allowed to write to (your own TNFS server, an SMB share, or the FujiNet's SD card are all good).

Go to BASIC — press **B** at the menu, or type **CAR** at the prompt, if a BASIC cartridge or built-in BASIC is waiting — and type:

COMPUTER: **READY**
YOU TYPE: **10 PRINT "HELLO NETWORK"** RETURN
YOU TYPE: **20 GOTO 10** RETURN

Now save it, exactly the way the 1050 manual taught:

YOU TYPE: **SAVE "D:MYFILE"** RETURN

The FujiNet's bus light flickers, and the program streams out of memory, through the FujiNet, and onto the server — because on this machine, **D:** is **N1:**. You could just as well have typed **SAVE "N1:MYFILE"** they land in the same place.

To prove the save worked, switch the computer off and on (boot NOS again), go to BASIC, and:

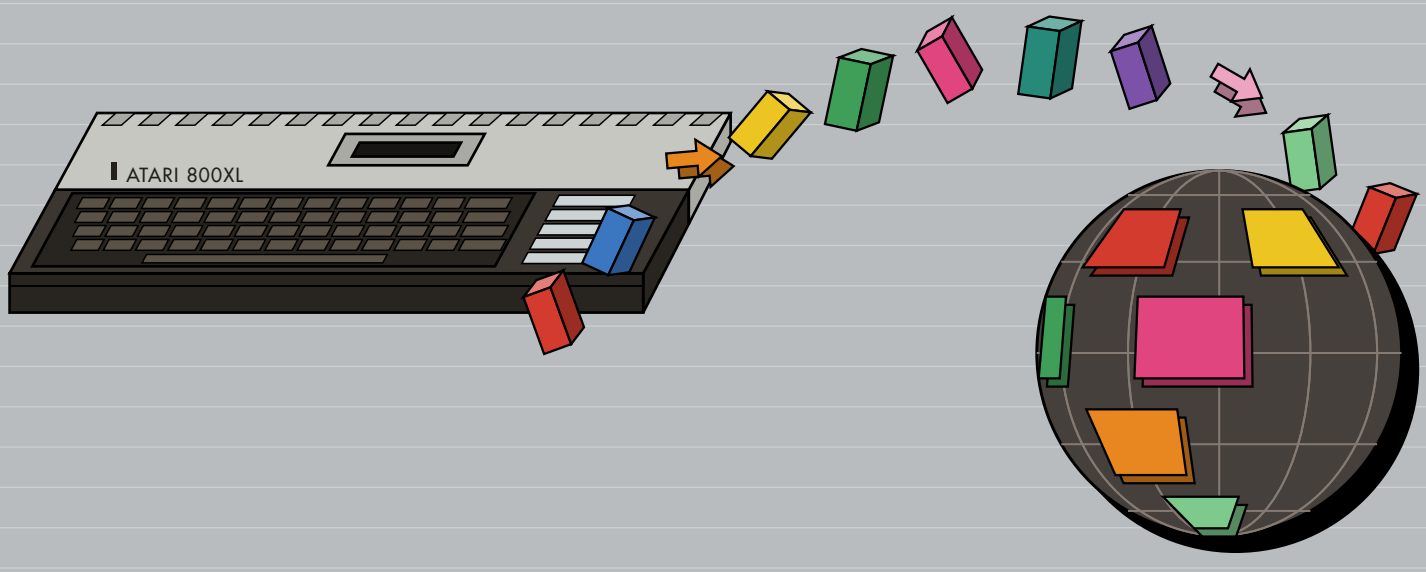
YOU TYPE: **LOAD "D:MYFILE"** RETURN
COMPUTER: **READY**
YOU TYPE: **LIST** RETURN

There is your program, back from across the network. **RUN** it and enjoy the applause. (Press BREAK to stop it.)

Everything else you know works the same way: **RUN "D:MYFILE"** loads and runs in one step; **LIST "D:PROG.LST"** and **ENTER "D:PROG.LST"** move listings; **PRINT#** and **INPUT#** tend data files.

NOTE AND POINT, TOO

Even BASIC's NOTE and POINT — the commands that jump around inside a file — work over the network now, wherever the protocol plays along. That story is new in NOS 1.0, and it gets its own chapter, "Moving Around In a File." Everything that reads or writes start-to-finish — which is nearly everything — is right at home already.



LOADING PROGRAMS

Most of the software you'll meet on the network is **binary** — machine-language programs with extenders like **.XEX** or **.COM**. The **LOAD** command brings one in and runs it — item **L. BINARY LOAD** on the menu, with **X** as a one-letter alias for the impatient at the prompt:

YOU TYPE: **LOAD JUMPMAN.XEX** RETURN

YOU TYPE: **X N2:GAMES/JUMPMAN.XEX**

LOAD understands the standard ATARI binary format — it follows the file's own instructions about where to sit in memory, runs any initialization on the way, and jumps to the program at the end. If a file isn't binary at all, NOS says so:

COMPUTER: **NOT A BINARY FILE**

Line-ending translation (see "Text Files") is switched off automatically on the drive doing the loading, so a binary never arrives scrambled. And loading is quick: whenever at least 128 bytes are on the way, NOS moves them in single bursts of up to 8K, streamed straight into their place in memory — no bucket brigade, byte by byte. "Inside NOS" tells how.

THE SHORTCUT

Type a bare word NOS doesn't recognize, and NOS assumes you mean a program: it adds **.COM** and tries to LOAD it from the current drive. If your server has an **ATARIWRITER.COM**, then this is all it takes:

YOU TYPE: **ATARIWRITER** RETURN

In effect, every **.COM** on the current drive is a NOS command.

COMING BACK

Some programs offer an "exit to DOS." With NOS underneath, that lands you back at the menu — and because NOS runs its commands from a buffer well below your program's quarters, you can often hop back in:

YOU TYPE: **REENTER** RETURN

REENTER (or **REE**) jumps to the program's own restart address. Two honest cautions. The menu you land on borrows the memory at $2600–$2AFF while it draws, so a program occupying that particular patch won't survive the round trip. And whether **any** program survives is the program's business — some re-initialize and wipe their slate — so save your work first.

You can also run machine code anywhere in memory by address — item **M** on the menu, **RUN** at the prompt (four hex digits, no **$**, lead small addresses with a zero):

YOU TYPE: **RUN A000** RETURN

YOU TYPE: **RUN 0600** RETURN

CARTRIDGES AND BASIC

CAR — item **B** on the menu — hands control to the cartridge (or built-in BASIC) without a cold start — your BASIC listing is still there when you arrive. Type **DOS** in BASIC to come back to NOS, work a while, then **CAR** again. One caution for long programs: coming back draws the menu, which borrows the memory from $2600 up — room for about 2.8K of BASIC program above NOS's floor. Longer than that, SAVE before you visit.

On XL and XE machines, **BASIC ON** and **BASIC OFF** (item **H**) swap the built-in BASIC in and out of the memory map without the ritual of rebooting while holding OPTION. (The command answers to **ROM ON** / **ROM OFF** too, for machines whose banked ROM holds something other than BASIC.) While ROM occupies $A000–$BFFF, NOS keeps the screen border gray as a reminder; **BASIC ON** when it's already on simply behaves like CAR.

SAVING BINARIES

SAVE — item **K** — writes a memory range out in the same binary format, with optional init and run addresses:

YOU TYPE: **SAVE MYPROG,2000,2FFF**

YOU TYPE: **SAVE MYPROG,2000,2FFF,,2000**

The double comma skips the init address while supplying the run address — the reference pages have the full recipe.

COPYING FILES

The **NCOPY** command (alias **COPY**) makes a copy of a file — on the same drive, or clear across the world from one server to another. It's item **C** on the menu, asking `COPY-FROM,TO?` at the prompt, name the source, a comma, and the destination:

YOU TYPE: `NCOPY MYFILE,MYFILE2`
YOU TYPE: `NCOPY N1:GAME.XEX,N2:GAME.XEX`

That second command is quietly astonishing: the file streams down from one server and up to another, with your ATARI conducting. When the destination keeps the same name, you may shorten it to a bare drive — or any folder ending in `/` — or, new in 1.0, leave it off altogether: one lone argument means "copy it **here,** onto the current drive."

YOU TYPE: `NCOPY N1:GAME.XEX,N2:`
YOU TYPE: `NCOPY GAME.XEX,N2:GAMES/`
YOU TYPE: `COPY N2:GAMES/ GAME.XEX`

NOS refuses to copy a file exactly onto itself — `SAME FILE?` is its whole objection.

A HANDFUL AT ONCE

New in 1.0: give the source a wild card and NCOPY copies everything that matches, announcing each file as it goes — no questions asked:

YOU TYPE: `COPY *.BAS,N2:BACKUP/`
COMPUTER: `FILE1.BAS`
: `FILE2.BAS`
: `GAME.BAS`

ADDING ON

A third argument, `A`, appends the source to the end of the destination instead of replacing it — handy for building one big log out of many small pieces:

YOU TYPE: `NCOPY DAY2.TXT,LOG.TXT,A`

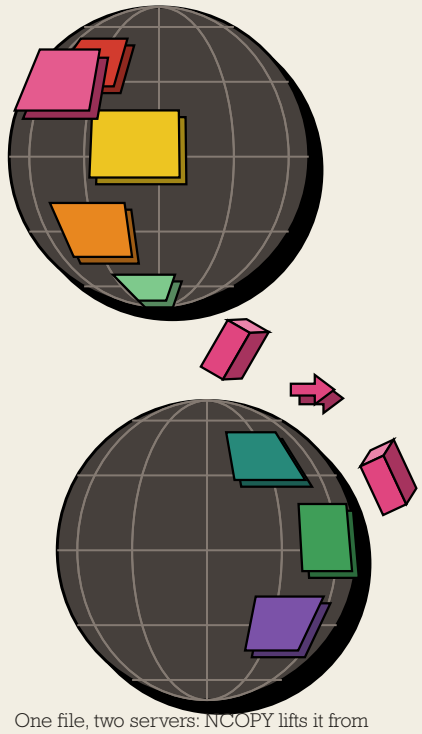
TWO CAUTIONS

- If line-ending translation is switched on (NTRANS — see "Text Files and Line Endings"), it alters what NCOPY carries — fine for text, ruinous for binaries. Leave translation at 0 when copying programs.
- NCOPY builds its network destination over drive `N4:` — another reason to keep your own mounts off drive 4.

COPYING TO DEVICES

The destination doesn't have to be a file. Send text to the printer, or straight to the screen:

YOU TYPE: `NCOPY NOTES.TXT,P:`
YOU TYPE: `NCOPY NOTES.TXT,E:`



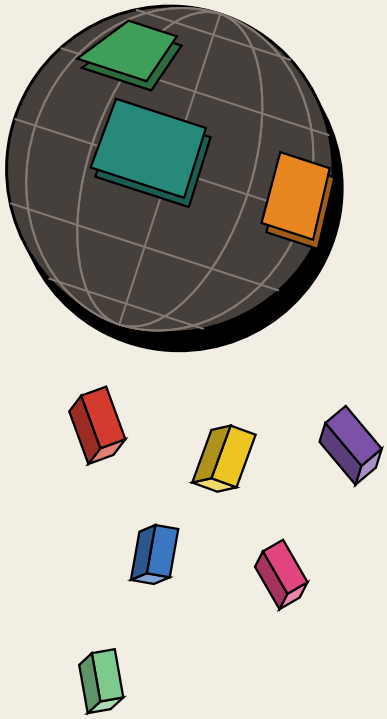
One file, two servers: NCOPY lifts it from the first volume and lays it onto the second.

DELETING AND RENAMING

The **DEL** command removes files from a mounted drive. (It answers to **ERASE** and **ERA** as well, and to item **D. DELETE FILE(S)** on the menu.)

YOU TYPE: `DEL MYFILE.AWP` RETURN
YOU TYPE: `DEL N2:DOCS/OLD.TXT`
YOU TYPE: `DEL "N2:My File.AWP"`

Mind three things. A plain name deletes **immediately** — no are-you-sure. It is **case-sensitive** — ask DIR for the exact name first. And there is no undelete anywhere on the network.



DEL tips the volume: the file's blocks tumble away for good. There is no getting them back.

DELETING A CROWD

New in 1.0: give DEL a wild card, and it turns careful. Every file that matches is offered back by name, and nothing goes until you say Y:

YOU TYPE: `DEL *.BAK` RETURN
COMPUTER: `DRAFT1.BAK (Y/N)?`
YOU TYPE: `Y` RETURN
COMPUTER: `DRAFT2.BAK (Y/N)?`
YOU TYPE: `N` RETURN
COMPUTER: `NOTES.BAK (Y/N)?`

Any answer but Y — an N, a bare RETURN — spares that file and moves along to the next. So `DEL *.*` is not the catastrophe it was on DOS 2.0: it's an interview.

The safety net has this shape on purpose: the **pattern** casts wide, and the **questions** keep you honest. If you've already made your peace, answering a column of Y's still beats typing a column of DELs.

RENAMING

RENAME (alias **REN**, item **E**) gives one file a new name: old name, comma, new name.

YOU TYPE: `RENAME Draft.txt,FINAL.TXT`
YOU TYPE: `REN N2:A.TXT,B.TXT`

Give the new name bare — no drive, no path. (Renaming through a relative path is a known rough edge in this version; NCD to the file's folder first.) Wild cards don't reach RENAME — one file at a time here.

MAKING ROOM

Directories are yours to create and remove — items **F** and **G**, where the protocol allows:

YOU TYPE: `MKDIR PROJECTS` RETURN
YOU TYPE: `RMDIR PROJECTS` RETURN

RMDIR only removes an **empty** directory — clean it out first.

TEXT FILES AND LINE ENDINGS

To read a text file without leaving NOS, use **TYPE** — item **O** on the menu:

YOU TYPE: `TYPE README.TXT` RETURN

YOU TYPE: `TYPE N3:HTTPS://fujinet.online/`

TYPE clears the screen and shows the file a screenful at a time. Press any key for the next page; press ESC to stop. It quietly copes with text from other computers — carriage returns and linefeeds both — so most files read clean with no ceremony. Point it only at **text:** TYPE trusts what it reads, and a big binary can overrun its buffer and scramble memory.

WHY LINE ENDINGS MATTER

Your ATARI ends a line of text with its own EOL character (155). Nearly every other computer ends lines with CR, LF, or both. A file is just bytes; if it moves between worlds untranslated, it reads as one endless line on one side or staircases on the other.

The **NTRANS** command tells a network drive how to translate **as bytes flow through it,** each direction:

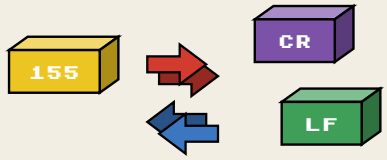
YOU TYPE: `NTRANS N1: 2` RETURN

MODE	TRANSLATION
0	none — bytes pass untouched
1	CR ⟷ ATARI EOL
2	LF ⟷ ATARI EOL (Unix, Mac)
3	CR/LF ⟷ ATARI EOL (Windows)

THE ONE RULE OF NTRANS

Translation is for text and **only** text. A binary program hauled through mode 2 arrives subtly broken — every byte that happened to equal 10 turned into 155. LOAD protects itself by switching translation off on its drive; NCOPY does not, so check before you copy programs.

Set it per drive and forget it: a drive mounted on a Unix or Windows server for text work wants mode 2 or 3; a drive full of ATARI software wants 0.



NTRANS swaps line endings in flight — ATARI's 155 on this side, CR and/or LF on that side.

MOVING AROUND IN A FILE

Most programs read a file the way you read a novel: front to back, no skipping. But some files are built for jumping around in — a database that hops straight to customer 500, a word processor that stitches pages out of order. On a diskette, BASIC's NOTE and POINT did the jumping. New in NOS 1.0, they jump across the network too.

NOTE asks **where am I?** POINT says **go there.** And the dialect is byte-counting, the way SpartaDOS speaks, not DOS 2's sector-and-offset: a position is simply how many bytes into the file you are, counted from zero.

```
YOU TYPE: 100 OPEN #1,4,0,"D:BIG.DAT"
YOU TYPE: 110 POINT #1,5000,0
YOU TYPE: 120 GET #1,B
```

Line 110 jumps clean to byte 5,000 — no reading past the first 4,999 to get there — and GET picks up the byte living there. The two numbers make one position: **position = first + 65,536 × second.** For files under 64K, which is most of them, the second number is simply 0, and the first is the byte position, plain as a page number.

Reading along, NOTE remembers a place worth returning to:

```
YOU TYPE: 200 NOTE #1,A,B
YOU TYPE: 210 REM READ ON A WHILE...
YOU TYPE: 220 POINT #1,A,B
```

NOS keeps the books straight behind the scenes: NOTE answers with **your program's** place — not the FujiNet's, which reads a little ahead — and POINT throws the read-ahead away and starts fresh at the new spot.

FROM MACHINE LANGUAGE

NOTE is XIO 38 and POINT is XIO 37, the same CIO commands every DOS answered. The position is 24 bits, riding in the IOCB's auxiliary bytes — ICAX3 low, ICAX4 middle, ICAX5 high — for a reach of 16 megabytes into any file. The source listing in the back shows both ends of the conversation (`PNOTE` and `PPOINT`).

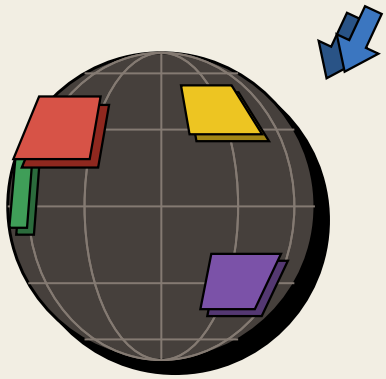
WHERE JUMPING WORKS

Jumping is the server's trick, so the protocol must play along:

PROTOCOL	NOTE AND POINT
TNFS, SD, SMB, NFS	jump anywhere, reading or writing
HTTP(S)	reading only
FTP, GDRIVE, streams	front to back only — no jumping

On the web the jump is a polite request to resume elsewhere — a **Range** request, the same trick download managers use — and the FujiNet quietly reopens the connection to make it. Fine for reading; there is no jumping while **writing** a web resource.

A protocol that can't jump answers POINT with `ERROR 166` — the same "invalid POINT" number DOS always reserved for the complaint.



POINT drops the needle anywhere on the volume; NOTE reads off exactly where it sits.

BATCH FILES

Any list of NOS commands can be saved as a text file and played back with one command. The file is called a **batch file,** and the command is **SUBMIT** — or just **@**.

YOU TYPE: `SUBMIT SETUP.BAT` RETURN

YOU TYPE: `@ SETUP.BAT` RETURN

Each line runs exactly as if you had typed it at the prompt. And because a batch file is only text, you can write it **anywhere** — in an ATARI editor, or on your PC in the comfort of a big screen, saved straight onto the server NOS reads it from. ATARI line endings, Unix LF, Windows CR/LF: SUBMIT reads them all as they come, no NTRANS needed.

THE BATCH TOOLKIT

Four commands exist mostly for batch files.

PRINT puts a message on the screen:

```
: PRINT "MOUNTING
  DRIVES..."
```

REM marks a comment — so does an apostrophe or **#** — and the line is ignored:

```
: REM SET UP MY
  MORNING
: ' THIS TOO IS A
  COMMENT
```

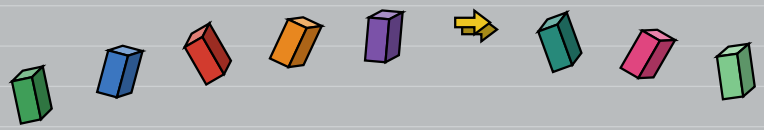
@SCREEN and **@NOSCREEN** control whether each command is echoed to the screen as it runs. A batch file runs **quietly** unless you ask: put `@SCREEN` where you want the play-by-play to start, `@NOSCREEN` where you want it to stop. (Lines that begin with `@` are never echoed.)

A MORNING BATCH FILE

Here is a `SETUP.BAT` that readies a whole desk:

```
REM -- MY MORNING SETUP --
PRINT "GOOD MORNING"
NCD N1:TNFS://192.168.1.20/
WORK/
NCD N2:TNFS://APPS.IRATA.
ONLINE/
NCD N3:SD://SCRATCH/
NTRANS N1: 2
PRINT "DRIVES 1-3 READY"
```

Drive 1 mounts your PC's work folder (with Unix line endings translated), drive 2 a public library, drive 3 the FujiNet's own SD card. The next section makes a file like this run **itself.**



STARTING UP YOUR WAY

Atari DOS had AUTORUN.SYS: put the right file on the right diskette, and the computer set itself up at boot. NOS has the same idea with a twist worth understanding: the setting lives **in your FujiNet,** not on any disk.

Tell NOS which batch file to run at boot by handing **AUTORUN** a full URL — protocol, host, path, and all, since at boot time nothing is mounted yet:

YOU TYPE:
```
AUTORUN
TNFS://192.168.1.20/
SETUP.BAT
```

From then on, every cold start ends with that batch file playing through — drives mounted, translation set, message on the screen, ready to work before you've touched a key.

WHERE THE SETTING LIVES

When you set AUTORUN, NOS writes the URL into an **AppKey** — a small named record the FujiNet keeps for programs — and AppKeys are stored on the microSD card in the FujiNet itself, in a file called `/FujiNet/db790000.key`. So the FujiNet needs an SD card (FAT32) for AUTORUN to stick.

Because the setting rides in the FujiNet:

- It survives power-off, and doesn't care which disk image booted.
- Carry your FujiNet to a friend's ATARI, and your startup comes along.
- Swap SD cards, and it stays with the **card.**

ASKING AND CLEARING

Query the current setting with a question mark; clear it with an empty string:

YOU TYPE: `AUTORUN ?` RETURN

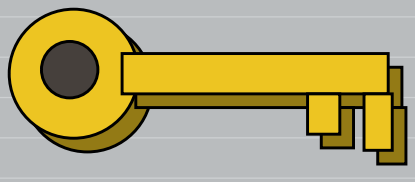
COMPUTER:
```
TNFS://192.168.1.20/
SETUP.BAT
```

YOU TYPE: `AUTORUN ""` RETURN

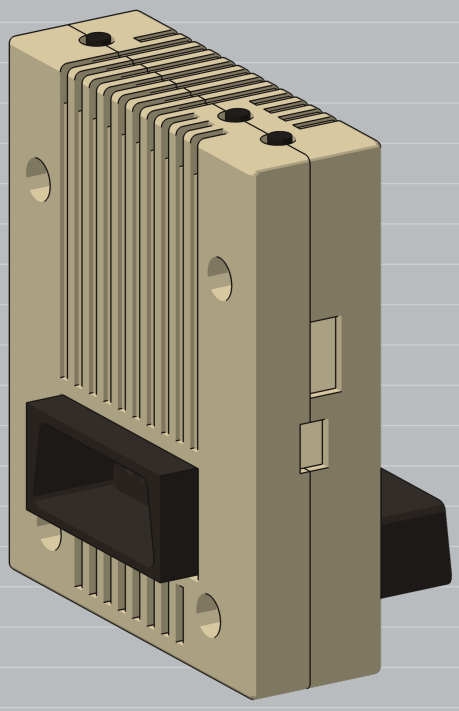
SKIPPING IT

Batch file gone wrong, or just want a plain prompt? Hold OPTION while NOS boots, and AUTORUN is skipped for that start.

Your startup, kept like a key in the FujiNet's pocket: an AppKey on its SD card, waiting for every cold start.



`db790000.key`



WHAT NOS DOESN'T DO

NOS wears DOS's clothes — the menu, the letters, the prompt — so it's fair to ask what it deliberately leaves out. The list is short, and every entry on it has the same explanation: **NOS contains no File Management System.** It cannot read or write diskettes or disk images — not even the ones your FujiNet mounts in its disk slots. `D:` goes to the network, full stop.

NO LOCK AND UNLOCK

Whether a network file may be changed is the **server's** decision, not NOS's. To protect files, set permissions on the server (or mount read-only media). A read-only file gives a write error, much like DOS's ERROR 167.

THE LAST FEW GAPS

- **RENAME** takes one plain name — the only file command wild cards haven't reached.
- There is no **MOVE** — copy, then delete.
- DIR lists names and sizes, but no dates — servers know them; NOS doesn't ask yet.

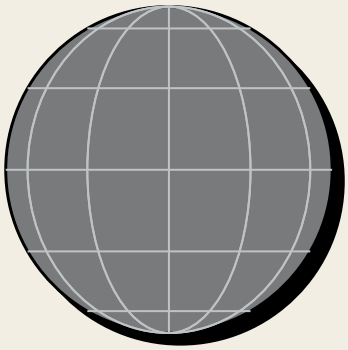
NO DISKETTES AT ALL

When you really need a diskette or an ATR disk image — yesterday's software library, a protected original, a disk-based application — use a disk operating system for disks and NOS for the network:

- Reboot into your FujiNet's CONFIG, mount the disk image, and boot it. Coming back to NOS is one more reboot.
- To move files between a diskette world and the network world, boot a classic DOS with the FujiNet `N:` handler loaded (from `n-handler.atr`) — there `D:` and `N:` exist side by side, and DOS's own copier moves files between them.
- Files on the FujiNet's SD card need no diskette at all: mount them directly with the `SD://` protocol.

THE PLEASANT SURPRISES

The ledger runs the other way too. NOS never asks you to swap diskettes mid-copy. It never runs out of room at 707 sectors. Its menu is ten small sectors, not a whole DUP.SYS — and MEM.SAV, that slow insurance policy, was never needed. Directories nest as deep as you please. Filenames breathe. And drive 2 can be in another country.



Formatting: the one chore the network never asks of you. A fresh mount arrives ready to use.

WHAT TO DO IF IT DOESN'T WORK

In most cases, when something goes wrong, NOS prints a message or an error number. You haven't broken anything. The common ones and their cures are listed here.

```
BOOT ERROR
BOOT ERROR
BOOT ERROR
```

BOOT ERROR

The computer can't find a bootable disk. Check that your FujiNet is on and connected, that **`NOS.ATR`** is mounted in **disk slot 1,** and that the slot is set to boot.

```
N1:LOAD GAME.XEX
170
N1:
```

ERROR 170: FILE NOT FOUND

The old classic, and the case-sensitive network gives it new life. DIR first; match the name letter for letter. You'll also meet it when a bare word isn't a command: NOS tried to fetch **`WORD.COM`** and found no such file. **`CMD?`** is its cousin — NOS couldn't even parse the line.

WHEN NOS ASKS A QUESTION

A one-line answer like **`FILE?`** or **`PATH?`** or **`Nn?`** isn't scolding — it's the command reminding you what it needs: a filename, a path, a drive between 1 and 8. **`MODE? 0=NONE, 1=CR, 2=LF, 3=CR/LF`** is NTRANS showing its whole menu.

ERROR 138: TIMEOUT

The computer called and nobody answered. The FujiNet is off, still starting up, or has lost the wireless network. Its status light tells the story.

ERROR 136: END OF FILE

Usually honest — the file simply ended — but arriving unexpectedly it means **the connection closed:** the server went away, or the mount was never made. NPWD the drive and look.

ERROR 146: NOT IMPLEMENTED

You asked a protocol for a trick it doesn't do — renaming on GDRIVE, writing to a plain web server, MKDIR on a read-only guest login. The protocol table in "The Protocols" is the map of what works where.

ERROR 144: THE SERVER SAID NO

DOS veterans know 144 as "device done error"; on the network it means the server refused — often a permissions problem, a read-only share, or a full disk on the far end. NOS asks the FujiNet for the specific reason and prints **that** number when it has one, so you may see a code from the server's world instead.

ERROR 165: BAD FILENAME

The filespec itself is malformed — a stray colon, a protocol misspelled, quotes forgotten around a name with spaces.

STILL STUCK?

NOS's online help is one command away — see the next page. The FujiNet community keeps a Discord where NOS's own authors answer questions, linked from **`fujinet.online`** and if you've found a genuine bug, the source listing in the back of this booklet is an invitation.

GETTING HELP

NOS carries a reference library it doesn't have room to hold: the **HELP** command fetches articles over the network, from the NOS project's own pages on GitHub, and shows them a screenful at a time like TYPE.

YOU TYPE: **HELP** RETURN

plain HELP lists the top-level topics. Ask for a topic to see what's inside it, and for **TOPIC/ARTICLE** to read one:

YOU TYPE: **HELP NOS** RETURN

YOU TYPE: **HELP NOS/MKDIR** RETURN

YOU TYPE: **HELP REF/ATASCII** RETURN

Any key turns the page; ESC closes the book. The next page of this booklet is the card catalog.

WHAT'S ON THE SHELF

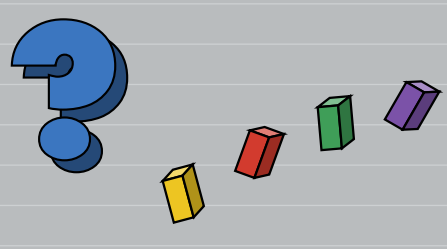
TOPIC	COVERS
NOS	every NOS command, one article each
MAP	the ATARI memory map — 1,082 cards, label by label
REF	reference tables: ATASCII, colors, key codes, error codes
ASM	a 6502 assembly reference, instruction by instruction
DEV	developer tools
UTL	utilities

A whole programmer's bookshelf — more than 1,200 articles — an arm's reach from the prompt, weighing nothing.

THE FINE PRINT

HELP needs the network, of course — it reads its articles over drive **N4:**, which is the main reason this booklet keeps advising you to leave drive 4 unmounted.

If HELP answers with an HTTP error like 404, the topic path wasn't quite right: articles under a topic need the topic in front — **HELP NOS/MKDIR** finds what **HELP MKDIR** cannot.



Ask the network; the network answers.

WHERE THE LIBRARY LIVES

Every article is a plain text file in the **fujinet-nhandler** repository on GitHub, under **nos/HELP/**. HELP fetches them from GitHub's raw pages the moment you ask, so what you read at the prompt is exactly what the repository holds today. Fix a typo, write a missing article, and every NOS in the world gets the improvement.

THE NOS SHELF

One card per command — ask as **HELP NOS/DIR**:

drives	**NCD NPWD Nn: NTRANS**
files	**DIR MKDIR RMDIR COPY NCOPY DEL RENAME TYPE**
programs	**CAR LOAD REENTER RUN SAVE**
batch	**SUBMIT AUTORUN REM @SCREEN @NOSCREEN**
odd jobs	**CLS PRINT BASIC DUMP FILL COLD WARM XEP HELP**

THE PROGRAMMER'S SHELVES

ASM is a 6502 reference: summary cards for the instruction set, address modes, branching, loops, and arithmetic — then one card per instruction, ADC through TYA.

YOU TYPE: **HELP ASM/INSTR**
YOU TYPE: **HELP ASM/LDA**

MAP is the whole ATARI memory map — 1,082 cards set from the text of **Mapping the ATARI** by Ian Chadwick. Browse it by letter, by label, or by address:

YOU TYPE: **HELP MAP/S**
YOU TYPE: **HELP MAP/SAVMSC**
YOU TYPE: **HELP MAP/0230**

A letter card lists every label under that letter with its address; the label and address cards tell you what the location does — the sort of question that once meant a book within arm's reach. Now the book **is** the arm's reach.

THE REFERENCE SHELF

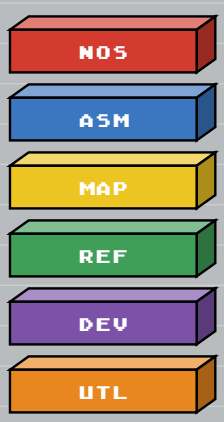
REF holds the tables taped inside every ATARI programmer's desk drawer: ATASCII codes, color values, keyboard codes, and the error numbers — with one card per error:

YOU TYPE: **HELP REF/ATASCII**
YOU TYPE: **HELP REF/ERROR/144**

DEV covers development tools (the ATARI Assembler Editor, to start), and **UTL** the utilities (the **T:EDIT** text editor).

ONE UNLISTED CARD

HELP BUGS is the project's standing confession: the known bugs, kept honestly, straight from the repository.



Six shelves, twelve hundred cards, and not one of them on your bookcase.

COMMAND REFERENCE

Every command NOS understands, in alphabetical order. Square brackets mark optional parts; `Nn:` means any drive name `N1:` through `N8:` (when omitted, the current drive is used). Commands may be typed in either case. A letter at an entry's top right names its item on the NOS menu.

@NOSCREEN

```
@NOSCREEN
```

Stops the echo of batch-file commands to the screen (the quiet state a batch file starts in). Counterpart of @SCREEN.

@SCREEN

```
@SCREEN
```

Starts echoing each batch-file command to the screen as it runs. Lines beginning with `@` are never echoed. Example: put `@SCREEN` at the top of a batch file to watch it work.

AUTORUN

```
AUTORUN PROTO://HOST[:PORT]/[PATH/]FILE
```

```
AUTORUN ?
```

```
AUTORUN ""
```

Names a batch file to SUBMIT automatically at every cold start. `?` shows the current setting; `""` clears it. Example: `AUTORUN TNFS://192.168.1.20/SETUP.BAT`

- The URL must be complete — nothing is mounted yet at boot.
- Stored as FujiNet AppKey db79/00/00: file `/FujiNet/db790000.key` on the FujiNet's (FAT32) SD card; 64 characters most.
- Hold OPTION during boot to skip the AUTORUN.

BASIC

MENU: H ALSO: ROM

```
BASIC ON|OFF
```

```
ROM ON|OFF
```

Switches the ROM at $A000–$BFFF (usually built-in BASIC) in or out of the memory map, without rebooting while holding OPTION. `BASIC ON` when ROM is already in behaves like CAR.

- XL/XE machines with built-in BASIC only — otherwise `NO BUILT-IN BASIC`.
- Performs a warmstart to settle the change.
- While ROM is switched in, NOS keeps the screen border gray as a reminder.

CAR

MENU: B

```
CAR
```

Leaves NOS for the cartridge (or built-in BASIC) through its warmstart vector — memory is preserved in both directions, so a BASIC listing survives the round trip. Type `DOS` there to come back.

- `NO CARTRIDGE` means nothing is there to run — no cartridge, and built-in BASIC switched out.

CLS

```
CLS
```

Clears the screen.

COLD

```
COLD
```

Reboots the computer (coldstart). On XL/XE machines, hold OPTION as it restarts to keep BASIC out.

DEL

MENU: D ALSO: ERASE, ERA

```
DEL [Nn:][PATH/]FILE
```

```
DEL [Nn:][PATH/]PATTERN
```

Removes one file, or — with a wild card — a confirmed crowd of them, from a mounted drive. Examples: `DEL N2:DOCS/OLDFILE.TXT` `DEL *.BAK`

- A plain name deletes immediately — no confirmation, no undelete.
- A pattern (`*`, `?`) offers each matched file with `(Y/N)?` — only Y deletes; matched names fill a 512-byte list per run.
- Names are case-sensitive; quote names with spaces: `DEL "N2:My File.AWP"`

DIR

MENU: A

```
DIR [Nn:][PATH/][PATTERN]
```

Lists files at the current spot (or the given path) with sizes — bytes, `K`, or `M` — and folders marked with a trailing `/`. Examples: `DIR` `DIR N2:*.XEX` `DIR GAMES/`

- `*` matches any run of characters, `?` exactly one.
- Directories are always listed, whatever the pattern.
- Hold SPACE to pause; ESC stops the listing.

DUMP

```
DUMP START [END]
```

Shows memory in hex, eight bytes per line, from START to END (or one line's worth if END is omitted). Example: `DUMP 0600 067F`

- Addresses are four hex digits, no dollar sign; lead small ones with zeros.
- ESC stops a long dump.

FILL

```
FILL START END XX
```

Fills memory from START through END with the byte XX (two hex digits). Example: `FILL 5000 5FFF 00` clears a 4K block.

HELP

```
HELP [TOPIC[/ARTICLE]]
```

The online manual. Plain HELP lists topics (NOS, MAP, REF, ASM, DEV, UTL); a topic lists its articles; topic/article shows the text, paged like TYPE.

- Articles arrive over drive `N4:` from the NOS project's GitHub pages; a mount on drive 4 can confuse it.
- Articles under a topic need the topic in the path: `HELP NOS/MKDIR`, not `HELP MKDIR`.

LOAD

MENU: L ALSO: X

```
LOAD [Nn:][PATH/]FILE
```

```
FILENAME
```

Loads a standard ATARI binary file (`.XEX`, `.COM`) and runs it, honoring the file's init and run addresses. Example: `LOAD N2:GAMES/JUMPMAN.XEX`

- A bare unrecognized word is treated as `LOAD WORD.COM` from the current drive.
- Line-ending translation is switched off automatically for the load.
- A non-binary file stops with `NOT A BINARY FILE`.

MENU

```
MENU
```

Returns from the command line to the NOS menu — the reverse of menu item P.

- The menu module is reloaded from the NOS disk image each time it draws — one more reason NOS.ATR stays mounted.

MKDIR

MENU: F

```
MKDIR [Nn:][PATH/]DIRNAME
```

Creates a directory on a mounted drive, where the protocol allows it. Example: `MKDIR PROJECTS`

NCD

MENU: I ALSO: CD, CWD

```
NCD [Nn:]PROTO://HOST[:PORT]/[PATH/]
```

```
NCD PATH | NCD ..
```

```
NCD Nn:
```

The mount command. With a full URL, connects a drive to a server; with a bare path (or `..`), moves around inside the mount; with a bare drive name, disconnects that drive.

- A trailing `/` is added if missing; quote paths containing spaces.
- No check is made that the new path exists — DIR after moving.
- Mounts live in the FujiNet and are shared with running programs; move drives thoughtfully.

NCOPY

MENU: C ALSO: COPY

```
NCOPY [Nn:][PATH/]FILE[,[Nn:][PATH/]FILE]
[,A]
```

```
NCOPY PATTERN,DEST | NCOPY FILE,P:
```

Copies a file — or a wild-card crowd of them — within a drive, between drives (even between different servers), or to a device.

- With no destination, copies onto the **current** drive, keeping the name. Copying a file onto itself is refused (`SAME FILE?`).
- A source pattern (`*`, `?`) copies every match, echoing each name — no confirmation; matched names fill a 512-byte list per run.
- `,A` appends to the destination instead of replacing it.
- A destination of `Nn:` alone, or ending in `/`, keeps the source's filename. Destinations `P:` (printer) and `E:` (screen) also work.
- Active NTRANS translation alters what is copied — set mode 0 for binaries. Network destinations are built over drive `N4:`.

Nn:

MENU: N

```
Nn:
```

Typed alone, makes drive **n** (1–8) the current drive; the prompt follows. No check is made that the drive is mounted. Example: `N3:`

NPWD

MENU: J ALSO: PWD

```
NPWD [Nn:]
```

Shows where a drive is mounted — the full URL, protocol and all. A blank answer means nothing is mounted there.

NTRANS

```
NTRANS [Nn:] MODE
```

Sets a drive's line-ending translation for text exchanged with other kinds of computer. Example: `NTRANS N1: 2` for a Unix-flavored server.

- Modes: 0 none, 1 CR, 2 LF, 3 CR/LF — each swapped with ATARI EOL (155) as bytes flow.
- Text only! Translation corrupts binaries. LOAD disables it for itself; NCOPY does not.

PASS

```
PASS PASSWORD
```

Supplies the password half of the credentials used for protocols that want a login (SMB shares, for instance). Give USER and PASS **before** the NCD that mounts.

PRINT

```
PRINT "STRING"
```

Shows a message on the screen — a batch file's voice. Example: `PRINT "DRIVES READY"`

REENTER

ALSO: REE

```
REENTER
```

Jumps back into the last loaded program through its run (or init) address. `NO ADDR IN INITAD OR RUNAD` means there's nothing to return to.

- Whether the program survives re-entry is up to the program — save work before quitting to NOS.

REM

ALSO: ', #

```
REM COMMENT
```

A comment — NOS ignores the line. For batch files.

RENAME

MENU: E ALSO: REN

```
RENAME [Nn:][PATH/]OLDNAME,NEWNAME
```

Gives a file (or directory) a new name on the same mount. Example: `RENAME AtariWriter.xex,ATARIWRITER.COM`

- Give NEWNAME bare — no drive or path in the second half.
- One file at a time — wild cards don't reach RENAME.
- Known rough edge: renaming through a relative path misfires in this version; NCD to the file's directory first.

RMDIR

MENU: G

```
RMDIR [Nn:][PATH/]DIRNAME
```

Removes an **empty** directory, where the protocol allows.

RUN

MENU: M

```
RUN ADDR
```

Calls machine code at a hex address (four digits, no dollar sign — `RUN 0600`, not `RUN $600`).

SAVE

MENU: K

```
SAVE [Nn:]FILE,START,END[,INIT][,RUN]
```

Writes a memory range as a standard ATARI binary file, with optional init and run addresses for LOAD to honor later. Example: `SAVE N1:FONT.BIN,3800,3BFF`

- All addresses four hex digits. END is inclusive.
- Skip INIT but give RUN with a double comma: `SAVE PROG,2000,2FFF,,2000`

SUBMIT

ALSO: @

```
SUBMIT [Nn:][PATH/]FILE
```

Runs the NOS commands in a text file, line by line, as if typed. Example: `@ SETUP.BAT`

- Accepts ATARI, Unix (LF), and Windows (CR/LF) line endings as they come.
- Runs quietly unless the file says `@SCREEN`.

TYPE

MENU: O

```
TYPE [Nn:][PATH/]FILE
```

Shows a text file on the screen, a page at a time, coping with CR/LF along the way. Works on anything the drive can read — including a web page by URL.

- Any key shows the next screenful; ESC stops.
- Text only — a large binary can overrun the buffer and corrupt memory.

USER

```
USER NAME
```

Supplies the username half of the credentials for protocols that want a login. Pair with PASS, before the mount.

WARM

```
WARM
```

Warmstarts the computer (like pressing RESET).

XEP

```
XEP [40]
```

For the XEP80 80-column peripheral: **XEP** switches to the 80-column screen, **XEP 40** back to the normal one.

- Load the XEP80 handler first (for instance **LOAD XEP80HAN.COM**); use only with an XEP80 connected.

INSIDE NOS

This chapter is for the reader who wants to know how the watch ticks — and for the one who wants to add a gear. Everything here can be checked against the source listing that follows.

THE SHAPE OF MEMORY

At boot, NOS hooks the system's DOS vectors (DOSVEC and DOSINI, at $0A and $0C), installs its **N:** handler — answering for **D:** as well — and raises MEMLO to $1B00. That is the whole resident cost: **five kilobytes even,** from $0700 through $1AFF, less than DOS 2.0 asked with its default buffers. Everything from $1B00 up belongs to you.

The top of the resident block is working space. At $1900 sits **OVLBUF**, a 256-byte window — two disk sectors' worth — where overlay commands are brought in to run. Above it, two 128-byte buffers (**RBUF**, **TBUF**) smooth network reads and writes, and the command line itself lives down in page 5, at $0582.

Three tenants **borrow** your memory briefly, and only while their features are on duty. The menu is ten sectors read into $2600 each time it draws. The wild-card machinery keeps its scratch at $4000 and its code at $4300 while a pattern is being worked through. And NCOPY stages its 8K bursts in the free RAM just above MEMLO. None of them leave anything resident behind.

THE OVERLAY GAME

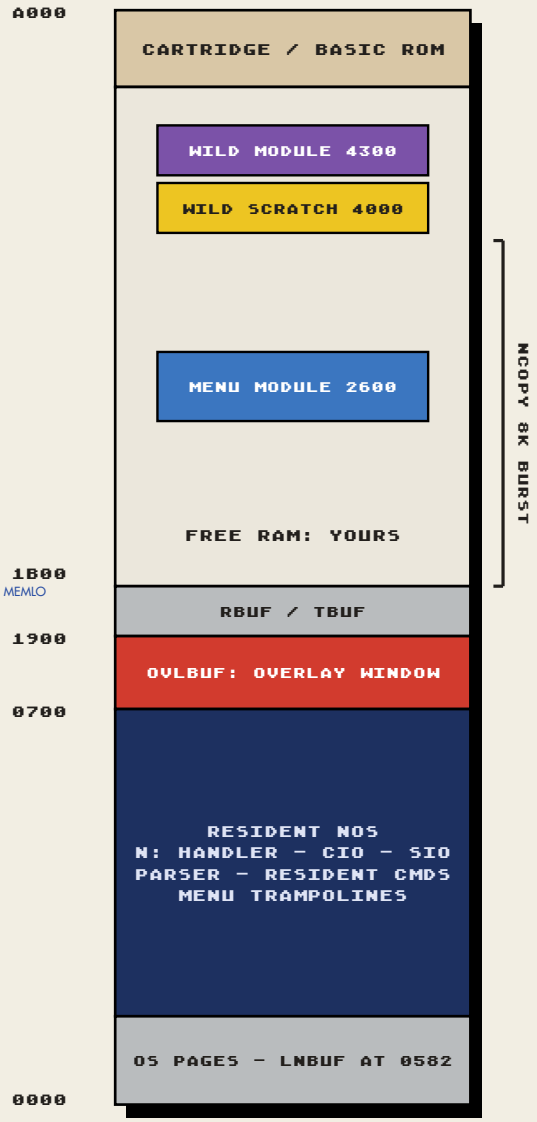
A command set this size won't fit in five kilobytes, and NOS solves it the way DOS always did: keep the everyday machinery resident and leave the rest on the disk. Two dozen commands run straight from the kernel. The bigger ones — AUTORUN, BASIC, DIR, DUMP, FILL, HELP, NCOPY, NTRANS, REENTER, SAVE, XEP, and DEL's wild-card scanner — live on **NOS.ATR** as **overlays,** one or two sectors each.

Calling one is plain arithmetic. Because the whole OS is assembled as one image whose ATR begins at $0700, a routine's assembled address **is** its place on the disk:

```
sector = address / 128 - 13
;  $0700/$80 - $0D = sector 1  (boot)
;  $2080/$80 - $0D = sector 52 (NCOPY)
```

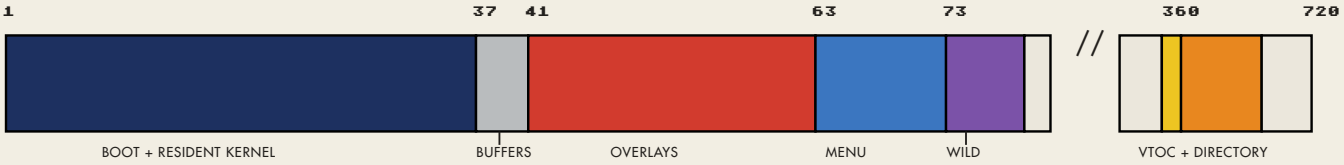
The resident dispatcher (**DO_OVERLAY**) looks up that sector in a table, reads the overlay into OVLBUF, and jumps to it — remembering what it loaded, so running DIR twice reads the disk once. An overlay executes at $1900 though it was assembled at its disk address, so its branches are relative and any absolute self-reference is spelled with one idiom: **OVLBUF-OVL_FOO+label**. A command too big for the window chains: NCOPY is three overlays that hand the copy from parser to opener to mover.

The menu and the wild-card engine are the second pattern: **transient modules,** assembled at their own run addresses ($2600 and $4300), loaded above MEMLO on demand, and expected to be stepped on — which is why menu picks are dispatched from resident trampolines that reload the module afterward.



The lay of memory under NOS 1.0 — resident kernel below MEMLO at $1B00, and three well-mannered tenants that borrow the free RAM only while they work. (Heights not to scale.)

THE DISK THEY RIDE ON



NOS.ATR is one long act of arithmetic. Sectors 1–36 boot and hold the resident kernel; 37–40 are the images of its buffers; 41–62 carry the command overlays, one or two sectors apiece, each at the address the formula predicts; 63–72 are the menu module and 73–78 the wild-card module. Way out at sector 360 sits a **fake** VTOC, with a hand-built directory in 361–368 whose entry names spell out the OS's own name — so that disk tools examining the image see a healthy single-density diskette instead of fainting. Nothing else on the disk is real: there is no File Management System in NOS, not even for its own disk.

THE BURST ENGINE

Ordinarily the **N:** handler hands bytes through a 128-byte buffer, the classic way. But on a **binary** read of 128 bytes or more, with data waiting, NOS skips the bucket brigade: one SIO frame carries **min(bytes waiting, buffer, 8K)** straight into the caller's memory — the exact count requested, nothing copied twice. Writes mirror it. The last byte of each burst rides through CIO so the bookkeeping stays honest, and the status poll keeps the true byte count on the side while the classic path still sees the old 127-byte ceiling. This is what makes NCOPY and LOAD feel like a disk that spins at wireless speed.

ROLLING YOUR OWN OVERLAY

To add a command **FOO**, you touch four tables and write one body — the sectors take care of themselves.

```
; 1. name it: CMD_IDX enum + keyword table
        FOO             ; in .ENUM CMD_IDX
        .CB "FOO"        ; keyword
        .BYTE CMD_IDX.FOO
; and CMD_TAB_L/H entries -> DO_FOO-1
```

```
; 2. a resident stub
DO_FOO: LDX #OVL_IDX.FOO
        JMP DO_OVERLAY
```

```
; 3. overlay tables: where + how many
 .BYTE <(OVL_FOO/SECTOR_SIZE-$0D)
 .BYTE [END_OVL_FOO-OVL_FOO]/SECTOR_SIZE
```

```
; 4. the body, sector-aligned
OVL_FOO:
        ; runs at OVLBUF ($1900)!
        ; branches: relative only
        ; absolute self-references:
        ;   OVLBUF-OVL_FOO+label
        .ALIGN SECTOR_SIZE,$00
END_OVL_FOO:
```

Arguments arrive pre-chewed: **CMDSEP** holds offsets into the line buffer for each argument, commas already split. The kernel lends its tools — **PREPEND_DRIVE** to default the drive, **DOSIOV** to talk SIO, **CIOOPEN**/**CIOGET**/**CIOPUT**/**CIOCLOSE** for files, **PRINT_STRING** and **PRINT_ERROR** to speak. Outgrow two sectors and you chain, as NCOPY does. Then:

```
$ make
; MADS reassembles NOS.ATR; every
; sector number recomputes itself.
```

Patches are welcome — the source, like the HELP library, lives in **fujinet-nhandler** on GitHub.

APPENDIX: THE NOS SOURCE LISTING

For the advanced user, the curious, and the future contributor: the complete assembly source of NOS v1.0.0, **nos.s**, exactly as it builds into **NOS.ATR**. It assembles with MADS, and lives — with its HELP articles, tools, and history — in the **fujinet-nhandler** repository on GitHub under **nos/**. Patches are welcome; so are readers — and the chapter "Inside NOS" is the map to carry into these woods. The resident kernel (boot, the **N:** handler, the command processor) comes first; the overlay commands follow, each aligned to the sector it occupies on the disk; the menu and wild-card modules and the fake directory bring up the rear.

```
        ;; Compile with MADS

        ;; Authors: Thomas Cherryhomes
        ;;   <thom.cherryhomes@gmail.com>

        ;; Michael Sternberg
        ;;   <mhsternberg@gmail.com>

        ;; Optimizations being done by djaybee!
        ;; Thank you so much!

;.DEF BURST_MODE


DOSVEC  =   $0A         ; DOSVEC
DOSINI  =   $0C         ; DOSINI

        ;; CURRENT IOCB IN ZERO PAGE
ZIOCB   =   $20         ; ZP IOCB
ZICHID  =   ZIOCB       ; ID
ZICDNO  =   ZIOCB+1     ; UNIT #
ZICCOM  =   ZIOCB+2     ; COMMAND
ZICSTA  =   ZIOCB+3     ; STATUS
ZICBAL  =   ZIOCB+4     ; BUF ADR LOW
ZICBAH  =   ZIOCB+5     ; BUF ADR HIGH
ZICPTL  =   ZIOCB+6     ; PUT ADDR L
ZICPTH  =   ZIOCB+7     ; PUT ADDR H
ZICBLL  =   ZIOCB+8     ; BUF LEN LOW
ZICBLH  =   ZIOCB+9     ; BUF LEN HIGH
ZICAX1  =   ZIOCB+10    ; AUX 1
ZICAX2  =   ZIOCB+11    ; AUX 2
ZICAX3  =   ZIOCB+12    ; AUX 3
ZICAX4  =   ZIOCB+13    ; AUX 4
ZICAX5  =   ZIOCB+14    ; AUX 5
ZICAX6  =   ZIOCB+15    ; AUX 6

        ;; CIO only copies the first 12 IOCB bytes to/from the ZP
        ;; IOCB; $2C-$2F are CIO work bytes, so AUX3-5 must be read
        ;; and written in the real IOCB, indexed by ICIDNO.

ICIDNO  =   $2E         ; CIO: IOCB # * 16

LMARGN  =   $52         ; Left margin
FR0     =   $D4         ; Floating Point register 0 (used during Hex->ASCII conversion)
CIX     =   $F2         ; Inbuff cursor
INBUFF  =   $F3         ; Ptr to input buffer ($0580)
MAX_APPKEY_LEN = $40    ; Used with appkey files
SECTOR_SIZE = $80;

;----------------------------------------
; INTERRUPT VECTORS
; AND OTHER PAGE 2 VARS
;----------------------------------------

VPRCED  =   $0202       ; PROCEED VCTR
COLOR0  =   $02C4       ;
COLOR1  =   $02C5       ;
COLOR2  =   $02C6       ; MODEF BKG C
COLOR3  =   $02C7       ;
COLOR4  =   $02C8       ;
RUNAD   =   $02E0       ; RUN ADDRESS
INITAD  =   $02E2       ; INIT ADDRESS
MEMLO   =   $02E7       ; MEM LO
DVSTAT  =   $02EA       ; 4 BYTE STATS

;----------------------------------------
; PAGE 3
; DEVICE CONTROL BLOCK (DCB)
;----------------------------------------

DCB     =   $0300       ; BASE
DDEVIC  =   DCB         ; DEVICE #
DUNIT   =   DCB+1       ; UNIT #
DCOMND  =   DCB+2       ; COMMAND
DSTATS  =   DCB+3       ; STATUS/DIR
DBUFL   =   DCB+4       ; BUF ADR L
DBUFH   =   DCB+5       ; BUF ADR H
DTIMLO  =   DCB+6       ; TIMEOUT (S)
DRSVD   =   DCB+7       ; NOT USED
DBYTL   =   DCB+8       ; BUF LEN L
DBYTH   =   DCB+9       ; BUF LEN H
DAUXL   =   DCB+10      ; AUX BYTE L
DAUXH   =   DCB+11      ; AUX BYTE H

HATABS  =   $031A       ; HANDLER TBL

;----------------------------------------
; IOCB'S * 8
;----------------------------------------

IOCB    =   $0340       ; IOCB BASE
ICHID   =   IOCB        ; ID
ICDNO   =   IOCB+1      ; UNIT #
ICCOM   =   IOCB+2      ; COMMAND
ICSTA   =   IOCB+3      ; STATUS
ICBAL   =   IOCB+4      ; BUF ADR LOW
ICBAH   =   IOCB+5      ; BUF ADR HIGH
ICPTL   =   IOCB+6      ; PUT ADDR L
ICPTH   =   IOCB+7      ; PUT ADDR H
ICBLL   =   IOCB+8      ; BUF LEN LOW
ICBLH   =   IOCB+9      ; BUF LEN HIGH
ICAX1   =   IOCB+10     ; AUX 1
ICAX2   =   IOCB+11     ; AUX 2
ICAX3   =   IOCB+12     ; AUX 3
ICAX4   =   IOCB+13     ; AUX 4
ICAX5   =   IOCB+14     ; AUX 5
ICAX6   =   IOCB+15     ; AUX 6

ROWCRS  =   $0054
RAMTOP  =   $006A
SCRFLG  =   $02BB       ; Scroll flag
CH      =   $02FC       ; Hardware code for last key pressed
CH1     =   $02F2       ; Prior keyboard character code
BASICF  =   $03F8
LNBUF   =   $0582       ; Line Buffer (128 bytes)
;LNBUF   =   $1880       ; Line Buffer (128 bytes)

;----------------------------------------
; HARDWARE REGISTERS
;----------------------------------------

CONSOL  =   $D01F       ; Console switches
PORTB   =   $D301       ; On XL/XE, used to enable/disable BASIC
PACTL   =   $D302       ; PIA CTRL A

;----------------------------------------
; MATH PACK VECTORS
;----------------------------------------
FASC    =   $D8E6       ; Floating point to ASCII
IFP     =   $D9AA       ; Integer to floating point

;----------------------------------------
; OS ROM VECTORS
;----------------------------------------

CIOV    =   $E456       ; CIO ENTRY
SIOV    =   $E459       ; SIO ENTRY
WARMSV  =   $E474       ; Warmstart entry point
COLDSV  =   $E477       ; Coldstart entry point

;----------------------------------------
; CONSTANTS
;----------------------------------------

GETREC  =   $05         ; CIO CMD TO GET RECORD
PUTREC  =   $09         ; CIO CMD TO PUT RECORD
PUTCHR  =   $0B         ; CIO CMD TO PUT CHAR

DEVIDN  =   $71         ; SIO DEVID
DSREAD  =   $40         ; FUJI->ATARI
DSWRIT  =   $80         ; ATARI->FUJI
MAXDEV  =   4           ; # OF N: DEVS
EOF     =   $88         ; ERROR 136

EOL     =   $9B         ; EOL CHAR
CR      =   $0D         ; Carrige Return
LF      =   $0A         ; Linefeed

; Burst mode parameters
MINBURST =  $80         ; Min user buf len to burst (128)
MAXBURST =  $2000       ; Max bytes per burst frame (8192).
                        ; LO byte MUST be 0 (see CAPCHK).

OPTION  =   $03
ESC_KEY =   $1C         ; Hardware code for ESC
SPC_KEY =   $21         ; Hardware code for SPACE

OINPUT  =   $04         ; CIO/SIO direction
OOUTPUT =   $08         ; CIO/SIO direction
BOGUS   =   $F0         ; Bogus FujiNet SIO command byte

;ROM_BORDER = $92        ; Border color when program in ROM
ROM_BORDER = $06        ; Border color when program in ROM

; FujiNet SIO command bytes.
CMD_DRIVE_CHG       = $01
CMD_CD              = $2C
CMD_DIR             = $02
CMD_DEL             = $21
CMD_DUMP            = BOGUS
CMD_FILL            = BOGUS
CMD_LOAD            = $28
CMD_MKDIR           = $2A
CMD_NCOPY           = BOGUS
CMD_NPWD            = $30
CMD_NTRANS          = 'T'
CMD_PASS            = $FE
CMD_RENAME          = $20
CMD_RMDIR           = $2B
CMD_SAVE            = BOGUS
CMD_SUBMIT          = BOGUS
CMD_TYPE            = BOGUS
CMD_USER            = $FD
CMD_UNLOCK          = $24
CMD_CAR             = BOGUS
CMD_CLS             = BOGUS
CMD_COLD            = BOGUS
CMD_HELP            = BOGUS
CMD_BASIC           = BOGUS
CMD_NOSCREEN        = BOGUS
CMD_PRINT           = BOGUS
CMD_REENTER         = BOGUS
CMD_REM             = BOGUS
CMD_RUN             = BOGUS
CMD_SCREEN          = BOGUS
CMD_WARM            = BOGUS
CMD_XEP             = BOGUS
CMD_AUTORUN         = BOGUS
CMD_MENU            = BOGUS
```

```
;;; Macros ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        .MACRO DCBC
        .LOCAL
        LDY     #$0C
?DCBL   LDA     %%1,Y
        STA     DCB,Y
        DEY
        BPL     ?DCBL
        .ENDL
        .ENDM

; ATR Header
        ORG     $06f0
        OPT     h-
        DTA     $96,$02,$80,$16,$80
:11     DTA     $00

;;; Initialization ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

HDR:    .BYTE   $00                ; BLFAG: Boot flag equals zero (unused)
        .BYTE   [BOOTEND-HDR]/128  ; BRCNT: Number of consecutive sectors to read
        .WORD   HDR                ; BLDADR: Boot sector load address ($700).
        .WORD   $E4C0              ; BIWTARR: Init addr (addr of RTS in ROM)

        JMP     START

START:  LDA     DOSINI
        STA     RESET+1
        LDA     DOSINI+1
        STA     RESET+2

        LDA     #<RESET
        STA     DOSINI
        LDA     #>RESET
        STA     DOSINI+1
        LDA     #<DOS       ; Point to DOS & CP below
        STA     DOSVEC
        LDA     #>DOS
        STA     DOSVEC+1

        JMP     ALTMEML     ; Alter MEMLO

RESET:  JSR     $FFFF       ; Jump to extant DOSINI

        LDA     #'N'
        STA     RBUF
        JSR     IHTBS       ; Insert into HATABS

        LDA     #'D'        ; Redirect calls for D: to the
        STA     RBUF        ; N: handler for compatibility
        JSR     IHTBS       ; with software that assumes D:

;---------------------------------------
;  Alter MEMLO
;---------------------------------------
ALTMEML:
        LDA     #<PGEND
        STA     MEMLO
        LDA     #>PGEND
        STA     MEMLO+1

        ;; Back to DOS

        RTS

;---------------------------------------
; Insert entry into HATABS
;---------------------------------------

IHTBS:
        LDY     #$00
IH1     LDA     HATABS,Y
        BEQ     HFND
        CMP     RBUF        ; RBUF contains 'N' or 'D'
        BEQ     HFND
        INY
        INY
        INY
        CPY     #11*3
        BCC     IH1

        ;; Found a slot

HFND:
        LDA     RBUF        ; RBUF contains 'N' or 'D'
        TAX
        STA     HATABS,Y
        LDA     #<CIOHND
        STA     HATABS+1,Y
        LDA     #>CIOHND
        STA     HATABS+2,Y

        CPX     #'N'        ; Return if 'N' and we'll be back
        BNE     HATABS_CONT ; one more time for 'D'
        RTS

HATABS_CONT:
        ;; And we're done with HATABS

        ;; Query FUJINET

        JSR     STPOLL

        ;; Output Ready/Error

OBANR:
        LDA     DSTATS      ; Check DSTATS
        BPL     OBRDY       ; < 128 = Ready

        ;; Status returned error.

OBERR:
        LDA     #<BERROR
        LDY     #>BERROR
        BVC     OBCIO

        ;; Status returned ready.

OBRDY:
        LDA     #<BREADY
        LDY     #>BREADY

OBCIO:
        JSR     PRINT_STRING

        ;; Vector in proceed interrupt

SPRCED:
        LDA     #<PRCVEC
        STA     VPRCED
        LDA     #>PRCVEC
        STA     VPRCED+1

        ;; And we are done, back to DOS.
        CLC
        RTS

;;; End Initialization Code ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Copy command's template DCB struct to OS's DCB struct (12 bytes)
DOSIOV:
        STA     DODCBL+1
        STY     DODCBL+2
;        LDY     #$0C
        LDY     #$0B
DODCBL  LDA     $FFFF,Y
        STA     DCB,Y
        DEY
        BPL     DODCBL

SIOVDST:
        JSR     SIOV
        LDY     DSTATS
        TYA
        RTS


;---------------------------------------
; CIO OPEN
;---------------------------------------

OPEN:
        ;; Prepare DCB

        JSR     GDIDX       ; Get Device ID in X (0-3)
        LDA     ZICDNO      ; IOCB UNIT # (1-4)
        STA     OPNDCB+1    ; Store in DUNIT
        LDA     ZICBAL      ; Get filename buffer
        STA     OPNDCB+4    ; stuff in DBUF
        LDA     ZICBAH      ; ...
        STA     OPNDCB+5    ; ...
        LDA     ZICAX1      ; Get desired AUX1/AUX2
        STA     OPNDCB+10   ; Save them, and store in DAUX1/DAUX2
        LDA     ZICAX2      ; ...
        STA     OPNDCB+11   ; ...

        ;;  Copy DCB template to DCB

        LDA     #<OPNDCB
        LDY     #>OPNDCB

        ;;  Send to #FujiNet

        JSR     DOSIOV

        ;; Return DSTATS; on a device ERROR (144) fetch extended

OPCERR:
        JSR     GETEXT      ; ERROR 144 -> extended error in Y

       ; RESET BUFFER LENGTH + OFFSET

OPDONE:
        LDA     #$01
        STA     TRIP
        JSR     GDIDX
        LDA     #$00
        STA     RLEN,X
        STA     TOFF,X
        STA     ROFF,X
        TYA
        RTS                 ; AY = ERROR

OPNDCB:
        .BYTE   DEVIDN  ; DDEVIC
        .BYTE   $FF     ; DUNIT
        .BYTE   'O'     ; DCOMND
        .BYTE   $80     ; DSTATS
        .BYTE   $FF     ; DBUFL
        .BYTE   $FF     ; DBUFH
        .BYTE   $FE       ; DTIMLO
        .BYTE   $00     ; DRESVD
        .BYTE   $00     ; DBYTL
        .BYTE   $01     ; DBYTH
        .BYTE   $FF     ; DAUX1
        .BYTE   $FF     ; DAUX2

; End CIO OPEN
;---------------------------------------

;---------------------------------------
; CIO CLOSE
;---------------------------------------

CLOSE:
        JSR     DIPRCD      ; Disable Interrupts
        JSR     GDIDX
        JSR     PFLUSH      ; Do a Put Flush if needed.

        LDA     ZICDNO      ; IOCB Unit #
        STA     CLODCB+1    ; to DCB...

        LDA     #<CLODCB
        LDY     #>CLODCB

        JMP     DOSIOV

CLODCB .BYTE    DEVIDN      ; DDEVIC
       .BYTE    $FF         ; DUNIT
       .BYTE    'C'         ; DCOMND
       .BYTE    $00         ; DSTATS
       .BYTE    $00         ; DBUFL
       .BYTE    $00         ; DBUFH
       .BYTE    $FE            ; DTIMLO
       .BYTE    $00         ; DRESVD
       .BYTE    $00         ; DBYTL
       .BYTE    $00         ; DBYTH
       .BYTE    $00         ; DAUX1
       .BYTE    $00         ; DAUX2

; End CIO CLOSE
;---------------------------------------

;---------------------------------------
; CIO GET
;---------------------------------------

GET:    JSR     GDIDX       ; IOCB UNIT #-1 into X
        LDA     RLEN,X      ; Get # of RX chars waiting
        BNE     GETDISC     ; LEN > 0?
```

```
        ;; If RX buffer is empty, get # of chars waiting...

        JSR     STPOLL      ; Status Poll

        ;; If a burst read is possible, do it directly into the
        ;; user's buffer instead of filling RBUF a byte at a time.

        JSR     BCHK        ; Burst eligible? (C clear = yes)
        BCS     GTNORM      ; No -> normal one-byte-at-a-time path
        JMP     BGET        ; Yes -> burst (returns straight to CIO)

GTNORM: JSR     GDIDX       ; IOCB UNIT -1 into X (because Poll trashes X)
        LDA     DVSTAT      ; # of bytes waiting (0-127)
        STA     RLEN,X      ; Store in RX Len
        BEQ     RETEOF

GETDO:  LDA     ZICDNO      ; Get IOCB UNIT #
        STA     GETDCB+DCB_IDX.DUNIT
        LDA     DVSTAT      ; # of bytes waiting
        STA     GETDCB+DCB_IDX.DBYTL
        STA     GETDCB+DCB_IDX.DAUX1

        LDA     #<GETDCB
        LDY     #>GETDCB

        JSR     DOSIOV

        ;; Clear the Receive buffer offset.

        JSR     GDIDX       ; IOCB UNIT #-1 into X
        LDA     #$00
        STA     ROFF,X

GETDISC:
        ;LDA     DVSTAT+2    ; Did we disconnect?
        LDA     DVSTAT+3    ; Did we disconnect?
        BNE     GETUPDP     ; nope, update the buffer cursor.

        ;; We disconnected, emit an EOF.

RETEOF:
        LDY     #EOF
        TYA
        RTS                 ; buh-bye.

GETUPDP:
        DEC     RLEN,X      ; Decrement RX length.
        LDY     ROFF,X      ; Get RX offset cursor.

        ;; Return Next char from appropriate RX buffer.

        LDA     RBUF,Y

        ;; Increment RX offset

GX:     INC     ROFF,X      ; Increment RX offset.
        TAY                 ; stuff returned val into Y temporarily.

        ;; If requested RX buffer is empty, reset TRIP.

        LDA     RLEN,X
        BNE     GETDONE
        STA     TRIP

        ;; Return byte back to CIO.

GETDONE:
        TYA                 ; Move returned val back.
        LDY     #$01        ; SUCCESS

        RTS                 ; DONE...

GETDCB:
       .BYTE    DEVIDN      ; DDEVIC
       .BYTE    $FF         ; DUNIT
       .BYTE    'R'         ; DCOMND
       .BYTE    $40         ; DSTATS
       .BYTE    <RBUF       ; DBUFL
       .BYTE    >RBUF       ; DBUFH
       .BYTE    $FE           ; DTIMLO
       .BYTE    $00         ; DRESVD
       .BYTE    $FF         ; DBYTL
       .BYTE    $00         ; DBYTH
       .BYTE    $FF         ; DAUX1
       .BYTE    $00         ; DAUX2

; End CIO GET
;--------------------------------------

;--------------------------------------
; CIO PUT
;--------------------------------------

PUT:    ;; Add to TX buffer.

        PHA                 ; Save byte (needed for normal path)
        JSR     GDIDX

        ;; Try to burst the whole write directly from the user's
        ;; buffer.  Only for binary PUT CHARACTERS with a large
        ;; enough length and nothing already buffered in TBUF.

        LDA     TOFF,X      ; Pending buffered bytes?
        BNE     PUTNRM      ; Yes -> normal path
        LDA     ZICCOM      ; CIO command
        AND     #$02        ; Binary PUT CHARACTERS?
        BEQ     PUTNRM      ; No (text/record) -> normal path
        LDA     ZICBLH      ; Buffer length hi
        BNE     PUTBST      ; >= 256 -> burst
        LDA     ZICBLL      ; Buffer length lo
        CMP     #MINBURST   ; >= MINBURST?
        BCC     PUTNRM      ; No -> normal path
PUTBST: PLA                 ; Discard byte (burst re-reads from buffer)
        JMP     BPUT        ; Burst write (returns straight to CIO)

PUTNRM: PLA                 ; Restore byte
        LDY     TOFF,X      ; GET TX cursor.
        STA     TBUF,Y      ; TX Buffer

POFF:   INC     TOFF,X      ; Increment TX cursor
        LDY     #$01        ; SUCCESSFUL

        ;; Do a PUT FLUSH if EOL or buffer full.

        CMP     #EOL        ; EOL?
        BEQ     FLUSH       ; FLUSH BUFFER
        JSR     GDIDX       ; GET OFFSET
        LDA     TOFF,X
        CMP     #$7F        ; LEN = $FF?
        BEQ     FLUSH       ; FLUSH BUFFER

        RTS

       ; FLUSH BUFFER, IF ASKED.

FLUSH:  JSR     PFLUSH      ; FLUSH BUFFER
        RTS

PFLUSH:

       ; CHECK CONNECTION, AND EOF
       ; IF DISCONNECTED.

        JSR     STPOLL      ; GET STATUS
        LDA     DVSTAT+3
        BEQ     RETEOF

PF1:    JSR     GDIDX       ; GET DEV X
        LDA     TOFF,X
        BNE     PF2
        JMP     PDONE

       ; FILL OUT DCB FOR PUT FLUSH

PF2:    LDA     ZICDNO
        STA     PUTDCB+DCB_IDX.DUNIT

       ; FINISH DCB AND DO SIOV

TBX:    LDA     TOFF,X
        STA     PUTDCB+DCB_IDX.DBYTL
        STA     PUTDCB+DCB_IDX.DAUX1

        LDA     #<PUTDCB
        LDY     #>PUTDCB
        JSR     DOSIOV

        ; CLEAR THE OFFSET CURSOR
        ; AND LENGTH

        JSR     GDIDX
        LDA     #$00
        STA     TOFF,X

PDONE:  LDY     #$01
        RTS

PUTDCB .BYTE    DEVIDN      ; DDEVIC
       .BYTE    $FF         ; DUNIT
       .BYTE    'W'         ; DCOMND
       .BYTE    $80         ; DSTATS
       .BYTE    $80         ; DBUFL
       .BYTE    >TBUF       ; DBUFH
       .BYTE    $FE         ; DTIMLO
       .BYTE    $00         ; DRESVD
       .BYTE    $FF         ; DBYTL
       .BYTE    $00         ; DBYTH
       .BYTE    $FF         ; DAUX1
       .BYTE    $00         ; DAUX2

; End CIO PUT
;--------------------------------------

;--------------------------------------
; CIO STATUS
;--------------------------------------

STATUS: JSR     ENPRCD      ; ENABLE PRCD
        JSR     GDIDX       ; GET DEVICE#
        LDA     RLEN,X      ; GET RLEN
        BNE     STSLEN      ; RLEN > 0?
        LDA     TRIP
        BNE     STTRI1      ; TRIP = 1?

        ; NO TRIP, RETURN SAVED LEN

STSLEN: LDA     RLEN,X      ; GET RLEN
        STA     DVSTAT      ; RET IN DVSTAT

        ; If you don't need to preserve Y then use it instead of A
        LDA     #$00
        STA     DVSTAT+1

        ; and INY here
        LDA     #$01
        STA     DVSTAT+2
        STA     DVSTAT+3

        BNE     STDONE

        ; DO POLL AND UPDATE RCV LEN

STTRI1: JSR     STPOLL      ; POLL FOR ST

        ; TRIP tracks whether data is waiting (NOT the poll's
        ; connection flag), and we must NOT set RLEN here: no data
        ; was read into RBUF yet (that is GET's job).

        LDA     DVSTAT      ; bytes waiting lo
        ORA     DVSTAT+1    ; | hi
        BNE     STDONE      ; something waiting -> leave TRIP set
        STA     TRIP        ; nothing waiting -> TRIP = 0 (A=0)

        ; RETURN CONNECTED? FLAG.

STDONE: LDA     DVSTAT+2
        LDY     #$01
        RTS

       ; ASK FUJINET FOR STATUS

STPOLL:
        LDA     ZICDNO      ; IOCB #
        STA     STADCB+1

        LDA     #<STADCB
        LDY     #>STADCB

        JSR     DOSIOV

        ;; Save the uncapped bytes-waiting count for burst mode
        ;; before it gets clamped to 127 below.

        LDA     DVSTAT
        STA     BWRAW
        LDA     DVSTAT+1
        STA     BWRAW+1

        ;; > 127 bytes? make it 127 bytes.

        LDA     DVSTAT+1
        BNE     STADJ
```

```
        LDA     DVSTAT
        BMI     STADJ
        BPL     STP2        ; <= 127 bytes (N clear, always taken)

STADJ   LDA     #$7F
        STA     DVSTAT
        LDA     #$00
        STA     DVSTAT+1

       ; A = CONNECTION STATUS

STP2:   LDA     DVSTAT+2
        RTS

STADCB: .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   'S'         ; DCOMND
        .BYTE   $40         ; DSTATS
        .BYTE   <DVSTAT     ; DBUFL
        .BYTE   >DVSTAT     ; DBUFH
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $04         ; DBYTL
        .BYTE   $00         ; DBYTH
        .BYTE   $00         ; DAUX1
        .BYTE   $00         ; DAUX2

; End CIO STATUS
;--------------------------------------

;--------------------------------------
; CIO SPECIAL
;--------------------------------------

SPEC:   ; HANDLE LOCAL COMMANDS.

        LDA     ZICCOM
        CMP     #$0F        ; 15 = FLUSH
        BNE     S25           ; NO.
        JSR     PFLUSH      ; DO FLUSH
        LDY     #$01        ; SUCCESS
        RTS

S25:        CMP        #$25            ; POINT
        BNE         S26           ; No.
        JMP         PPOINT            ; Do Point

S26:        CMP        #$26            ; NOTE
        BNE         S1            ; No.
        JMP         PNOTE             ; Do Note

S1:     CMP     #40         ; 40 = LOAD AND EXECUTE
        BEQ     S2          ; YES.
        JMP     S3          ; NO. SKIP OVER spec40

S2:     RTS
       ; HANDLE SIO COMMANDS.
       ; GET DSTATS FOR COMMAND
S3:
        LDA     ZICDNO
        STA     SPEDCB+DCB_IDX.DUNIT
        LDA     ZICCOM
        STA     SPEDCB+DCB_IDX.DAUX1

        LDA     #<SPEDCB
        LDY     #>SPEDCB
        JSR     DOSIOV

        BMI     :DSERR

       ; WE GOT A DSTATS INQUIRY
       ; IF $FF, THE COMMAND IS
       ; INVALID

DSOK:   LDA     INQDS
        CMP     #$FF        ; INVALID?
        BNE     DSGO        ; DO THE CMD
        LDY     #$92        ; UNIMP CMD
        TYA
DSERR:
        RTS

        ;; Do the special, since we want to pass in all the IOCB
        ;; Parameters to the DCB, This is being done long-hand.

DSGO:   LDA     ZICCOM
        PHA
        LDA     #$00
        PHA
        LDA     INQDS
        PHA
        LDA     #$01
        PHA
        LDA     ZICBAL
        PHA
        LDA     ZICAX1
        PHA
        LDA     ZICBAH
        PHA
        LDA     ZICAX2
        PHA
        LDY     #$03
DSGOL:
        PLA
        STA     DBYTL,Y
        PLA
        STA     DCOMND,Y
        DEY
        BPL     DSGOL

        JMP     SIOVDST

        ;; Return DSTATS in Y and A

SPEDCB  .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   $FF         ; DCOMND ; inq
        .BYTE   $40         ; DSTATS
        .BYTE   <INQDS      ; DBUFL
        .BYTE   >INQDS      ; DBUFH
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $01         ; DBYTL
        .BYTE   $00         ; DBYTH
        .BYTE   $FF         ; DAUX1
        .BYTE   $FF         ; DAUX2

; End CIO SPECIAL
;--------------------------------------

;--------------------------------------
; CIO NOTE/POINT
;--------------------------------------
; NOTE ($26) returns, and POINT ($25) sets, the 24-bit linear
; file position in ICAX3 (lo), ICAX4 (mid), ICAX5 (hi).

PNOTE:  JSR     PFLUSH      ; PUSH PENDING PUT BYTES FIRST

        LDA     ZICDNO      ; UNIT #
        STA     NOTDCB+DCB_IDX.DUNIT

        LDA     #<NOTDCB
        LDY     #>NOTDCB
        JSR     DOSIOV      ; 3-BYTE POSITION INTO NPBUF

        LDY     DSTATS
        CPY     #$01        ; SUCCESS?
        BEQ     PNOK
        JMP     GETEXT      ; NO -- ERROR 144 to extended code in Y

       ; FUJINET'S POSITION IS AHEAD OF THE USER'S BY WHATEVER
       ; IS STILL UNREAD IN RBUF; SUBTRACT IT.

PNOK:   JSR     GDIDX       ; UNIT INTO X
        SEC
        LDA     NPBUF
        SBC     RLEN,X
        STA     NPBUF
        LDA     NPBUF+1
        SBC     #$00
        STA     NPBUF+1
        LDA     NPBUF+2
        SBC     #$00
        STA     NPBUF+2

       ; STORE INTO THE CALLER'S IOCB (SEE ICIDNO NOTE UP TOP).

        LDX     ICIDNO      ; IOCB # * 16
        LDA     NPBUF
        STA     ICAX3,X
        LDA     NPBUF+1
        STA     ICAX4,X
        LDA     NPBUF+2
        STA     ICAX5,X
        LDY     #$01
        RTS

PPOINT: JSR     PFLUSH      ; PUSH PENDING PUT BYTES FIRST

        LDX     ICIDNO      ; IOCB # * 16
        LDA     ICAX3,X     ; POSITION (LO)
        STA     NPBUF
        LDA     ICAX4,X     ; POSITION (MID)
        STA     NPBUF+1
        LDA     ICAX5,X     ; POSITION (HI)
        STA     NPBUF+2

        LDA     ZICDNO      ; UNIT #
        STA     PNTDCB+DCB_IDX.DUNIT

        LDA     #<PNTDCB
        LDY     #>PNTDCB
        JSR     DOSIOV      ; SEND 3-BYTE POSITION

        LDY     DSTATS
        CPY     #$01        ; SUCCESS?
        BEQ     PPOK
        JMP     GETEXT      ; NO -- ERROR 144 to extended code in Y

       ; DROP STALE READ-AHEAD AND TRIP SO THE NEXT GET/STATUS
       ; POLLS FRESH DATA AT THE NEW POSITION.

PPOK:   JSR     GDIDX       ; UNIT INTO X
        LDA     #$00
        STA     RLEN,X
        STA     ROFF,X
        LDA     #$01
        STA     TRIP
        LDY     #$01
        RTS

NOTDCB: .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   $26         ; DCOMND ; NOTE (TELL)
        .BYTE   $40         ; DSTATS
        .BYTE   <NPBUF      ; DBUFL
        .BYTE   >NPBUF      ; DBUFH
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $03         ; DBYTL ; 3 BYTES
        .BYTE   $00         ; DBYTH
        .BYTE   $00         ; DAUX1
        .BYTE   $00         ; DAUX2

PNTDCB: .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   $25         ; DCOMND ; POINT (SEEK)
        .BYTE   $80         ; DSTATS
        .BYTE   <NPBUF      ; DBUFL
        .BYTE   >NPBUF      ; DBUFH
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $03         ; DBYTL ; 3 BYTES
        .BYTE   $00         ; DBYTH
        .BYTE   $00         ; DAUX1
        .BYTE   $00         ; DAUX2

; End CIO NOTE/POINT
;--------------------------------------

;--------------------------------------
; BURST MODE
;--------------------------------------
; Burst mode short-circuits CIO's one-byte-at-a-time GET/PUT loop.
; The OS keeps its running buffer pointer in ZICBAL and its remaining
; byte count in ZICBLL while transferring.  On a GET it stores the
; returned byte, then bumps ZICBAL and drops ZICBLL by one; on a PUT
; it reads the byte, then does the same.  So a handler can move a
; whole block by transferring directly to/from (ZICBAL) and advancing
; ZICBAL / shrinking ZICBLL by (block-1); CIO's own final +1/-1
; accounts for the last byte.  Mirrors Atari DOS 2.0S FMS burst I/O.

       ; Decide whether a GET can burst.  Call right after STPOLL,
       ; while BWRAW holds the uncapped bytes-waiting count and
       ; DVSTAT+3 holds the connection flag (0 = disconnected).
       ; Eligible = binary GET CHARACTERS, user buffer >= MINBURST,
       ; still connected, and something actually waiting.
       ; Returns C clear if eligible, C set otherwise.

BCHK:   LDA     ZICCOM      ; CIO command
        AND     #$02        ; Bit 1 set = CHARACTERS (binary)
        BEQ     BCHKNO      ; Clear = RECORD (text) -> no burst
```

```
        LDA     ZICBLH      ; Buffer length hi
        BNE     BCHKAV      ; >= 256 -> big enough
        LDA     ZICBLL      ; Buffer length lo
        CMP     #MINBURST   ; >= MINBURST?
        BCC     BCHKNO      ; No -> too small to bother
BCHKAV: LDA     DVSTAT+3    ; Connection flag (0 = disconnected)
        BEQ     BCHKNO      ; Disconnected -> let normal path emit EOF
        LDA     BWRAW       ; Bytes waiting lo
        ORA     BWRAW+1     ; | hi
        BEQ     BCHKNO      ; Nothing waiting -> let normal path run
        CLC                 ; Eligible
        RTS
BCHKNO: SEC                 ; Not eligible
        RTS

       ; BURST GET: read min(bytes-waiting, buffer length, MAXBURST)
       ; bytes in one SIO call straight into the user's buffer,
       ; advance ZICBAL/ZICBLL past all but the last byte, and hand
       ; that last byte back to CIO.  Entry: BWRAW = bytes waiting.

BGET:   ;; CHUNK = min(BWRAW, ZICBLL)
        LDA     BWRAW+1     ; waiting hi
        CMP     ZICBLH
        BCC     BGBW        ; waiting < len -> use waiting
        BNE     BGBL        ; waiting > len -> use len
        LDA     BWRAW       ; hi equal, compare lo
        CMP     ZICBLL
        BCC     BGBW        ; waiting < len -> use waiting
BGBL:   LDA     ZICBLL      ; use length (length <= waiting)
        STA     CHUNK
        LDA     ZICBLH
        STA     CHUNK+1
        JMP     BGCAP
BGBW:   LDA     BWRAW       ; use bytes waiting
        STA     CHUNK
        LDA     BWRAW+1
        STA     CHUNK+1
BGCAP:  JSR     CAPCHK      ; Cap CHUNK to MAXBURST

        ;; Fill in the burst read DCB
        LDA     ZICDNO
        STA     BRDCB+DCB_IDX.DUNIT
        LDA     ZICBAL
        STA     BRDCB+DCB_IDX.DBUFL     ; user buffer
        LDA     ZICBAH
        STA     BRDCB+DCB_IDX.DBUFH
        LDA     CHUNK
        STA     BRDCB+DCB_IDX.DBYTL
        STA     BRDCB+DCB_IDX.DAUX1     ; bytes to read
        LDA     CHUNK+1
        STA     BRDCB+DCB_IDX.DBYTH
        STA     BRDCB+DCB_IDX.DAUX2

        LDA     #<BRDCB
        LDY     #>BRDCB
        JSR     DOSIOV
        LDY     DSTATS
        CPY     #$01        ; Success?
        BNE     BGERR

        JSR     ADVBUF      ; Advance past all but the last byte

        ;; If we drained everything that was waiting, clear TRIP so a
        ;; later STATUS reflects an empty buffer (like GETUPDP does).
        LDA     CHUNK
        CMP     BWRAW
        BNE     BGRET
        LDA     CHUNK+1
        CMP     BWRAW+1
        BNE     BGRET
        LDA     #$00
        STA     TRIP
BGRET:  LDY     #$00
        LDA     (ZICBAL),Y  ; A = last byte, for CIO to store
        LDY     #$01        ; Success
        RTS

       ; SIO error during burst read; return the extended code
       ; on a device ERROR (144), else DSTATS as-is.
BGERR:  JMP     GETEXT

       ; BURST PUT: write the whole remaining buffer (up to MAXBURST)
       ; in one SIO call straight from the user's buffer.  The byte
       ; CIO handed us is the first byte of the block, re-sent from
       ; memory; CIO's final +1/-1 accounts for it.

BPUT:   JSR     STPOLL      ; Check connection first (like PFLUSH)
        LDA     DVSTAT+3
        BNE     BPGO        ; Connected -> proceed
        LDY     #EOF        ; Disconnected -> EOF
        TYA
        RTS
BPGO:   LDA     ZICBLL      ; CHUNK = remaining length
        STA     CHUNK
        LDA     ZICBLH
        STA     CHUNK+1
        JSR     CAPCHK      ; Cap CHUNK to MAXBURST

        LDA     ZICDNO
        STA     BWRCB+DCB_IDX.DUNIT
        LDA     ZICBAL
        STA     BWRCB+DCB_IDX.DBUFL     ; user buffer
        LDA     ZICBAH
        STA     BWRCB+DCB_IDX.DBUFH
        LDA     CHUNK
        STA     BWRCB+DCB_IDX.DBYTL
        STA     BWRCB+DCB_IDX.DAUX1     ; bytes to write
        LDA     CHUNK+1
        STA     BWRCB+DCB_IDX.DBYTH
        STA     BWRCB+DCB_IDX.DAUX2

        LDA     #<BWRCB
        LDY     #>BWRCB
        JSR     DOSIOV
        LDY     DSTATS
        CPY     #$01
        BNE     BPERR

        JSR     ADVBUF      ; Advance past all but the last byte
        LDY     #$01        ; Success
        RTS

BPERR:  JMP     GETEXT

       ; Advance CIO buffer by (CHUNK-1): ZICBAL += CHUNK-1 ;
       ; ZICBLL -= CHUNK-1.  Leaves CIO pointing at the last
       ; transferred byte with one count still to go.

ADVBUF: LDA     CHUNK       ; DECR = CHUNK - 1
        SEC
        SBC     #$01
        STA     DECR
        LDA     CHUNK+1
        SBC     #$00
        STA     DECR+1
        CLC                 ; ZICBAL += DECR
        LDA     ZICBAL
        ADC     DECR
        STA     ZICBAL
        LDA     ZICBAH
        ADC     DECR+1
        STA     ZICBAH
        SEC                 ; ZICBLL -= DECR
        LDA     ZICBLL
        SBC     DECR
        STA     ZICBLL
        LDA     ZICBLH
        SBC     DECR+1
        STA     ZICBLH
        RTS

       ; Cap CHUNK to MAXBURST.  Relies on MAXBURST's low byte being
       ; 0: once CHUNK's hi byte reaches MAXBURST's hi byte the value
       ; is already >= MAXBURST.

CAPCHK: LDA     CHUNK+1
        CMP     #>MAXBURST
        BCC     CAPRTS      ; hi < MAXBURST hi -> under cap
        LDA     #<MAXBURST  ; else clamp to MAXBURST
        STA     CHUNK
        LDA     #>MAXBURST
        STA     CHUNK+1
CAPRTS: RTS

       ; Burst read DCB.  DBUF/DBYT/DAUX set at run time by BGET.
BRDCB   .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   'R'         ; DCOMND
        .BYTE   DSREAD      ; DSTATS
        .BYTE   $FF         ; DBUFL (set at run time)
        .BYTE   $FF         ; DBUFH (set at run time)
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $FF         ; DBYTL (set at run time)
        .BYTE   $00         ; DBYTH (set at run time)
        .BYTE   $FF         ; DAUX1 (set at run time)
        .BYTE   $00         ; DAUX2 (set at run time)

       ; Burst write DCB (same, DCOMND 'W').
BWRCB   .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   'W'         ; DCOMND
        .BYTE   DSWRIT      ; DSTATS
        .BYTE   $FF         ; DBUFL (set at run time)
        .BYTE   $FF         ; DBUFH (set at run time)
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $FF         ; DBYTL (set at run time)
        .BYTE   $00         ; DBYTH (set at run time)
        .BYTE   $FF         ; DAUX1 (set at run time)
        .BYTE   $00         ; DAUX2 (set at run time)

; End Burst Mode
;--------------------------------------

;######################################
;#                                    #
;#             CIO Functions          #
;#                                    #
;######################################

;--------------------------------------
CIOCLOSE:
;--------------------------------------
    ; X must contain IOCB offset ($10,$20,..)
        LDA     #$0C            ; Close #1 first
        STA     ICCOM,X
        JMP     CIOV

;--------------------------------------
CIOOPEN:
;--------------------------------------
    ; Input:
    ; X = IOCB offset ($10,$20,..)
    ; Y = data direction (4=inp,8=out,12=i/o)
    ; INBUFF contains ICBAL/H (filename)
        LDA     #$03            ; 3 = CIO 'OPEN FILE'
        STA     ICCOM,X
        LDA     INBUFF          ; Pointer to filename
        STA     ICBAL,X
        LDA     INBUFF+1        ; Pointer to filename
        STA     ICBAH,X
        TYA                     ; Y contains 4, 8, 12, etc
        STA     ICAX1,X         ; Data direction
        LDA     #$00
        STA     ICAX2,X         ; Unused
        JMP     CIOV            ; Call CIO
;        JSR    PRINT_ERROR
;
;CIOOPEN_DONE:
;        RTS

;--------------------------------------
CIOSTATUS:
;--------------------------------------
        LDA     #$0D
        STA     ICCOM,X
        JSR     CIOV
        BPL     CIOSTATUS_DONE
        JSR     PRINT_ERROR

CIOSTATUS_DONE:
        RTS


;--------------------------------------
CIOGET:
;--------------------------------------
    ; Input:
    ; X = IOCB offset ($10,$20,..)
    ; A = ICBLL
    ; Y = ICBLH
    ; INBUFF contains ICBAL/H
        PHA                 ; Stash Buffer length Lo
        LDA     #$07        ; GET BYTES command
        STA     ICCOM,X
        LDA     INBUFF      ; Get buffer addr from pointer
        STA     ICBAL,X
        LDA     INBUFF+1
        STA     ICBAH,X
        PLA                 ; Retrieve Buffer length Lo
        STA     ICBLL,X
```

```
        TYA                 ; Get Buffer length Hi
        STA     ICBLH,X
        JSR     CIOV        ; Bon voyage
        BPL     CIOGET_DONE
;        JMP     PRINT_ERROR

CIOGET_DONE:
        RTS

;---------------------------------------
CIOPUT:
;---------------------------------------
    ; Input:
    ; X = IOCB offset ($10,$20,..)
    ; A = ICBLL
    ; Y = ICBLH
    ; INBUFF contains ICBAL/H
        PHA                 ; Stash Buffer length Lo
        LDA     #$0B        ; PUT BYTES command
        STA     ICCOM,X
        LDA     INBUFF      ; Get buffer addr from pointer
        STA     ICBAL,X
        LDA     INBUFF+1
        STA     ICBAH,X
        PLA                 ; Retrieve Buffer length Lo
        STA     ICBLL,X
        TYA                 ; Get Buffer length Hi
        STA     ICBLH,X
        JSR     CIOV        ; Bon voyage
        BPL     CIOPUT_DONE
;        JMP     PRINT_ERROR

CIOPUT_DONE:
        RTS

;;---------------------------------------
;CIOGETREC:
;;---------------------------------------
;    ; Input:
;    ; X = IOCB offset ($10,$20,..)
;    ; A = ICBLL
;    ; Y = ICBLH
;    ; INBUFF contains ICBAL/H
;        PHA                 ; Stash Buffer length Lo
;        LDA     #$05        ; GET RECORD command
;        STA     ICCOM,X
;        LDA     INBUFF      ; Get buffer addr from pointer
;        STA     ICBAL,X
;        LDA     INBUFF+1
;        STA     ICBAH,X
;        PLA                 ; Retrieve Buffer length Lo
;        STA     ICBLL,X
;        TYA                 ; Get Buffer length Hi
;        STA     ICBLH,X
;
;        JSR     CIOV        ; Bon voyage
;        BPL     CIOGETREC_DONE
;;        JMP     PRINT_ERROR
;
;CIOGETREC_DONE:
;        RTS

;#####################################
;#                                   #
;#          Utility Functions        #
;#                                   #
;#####################################
    ; ENABLE PROCEED INTERRUPT

ENPRCD: LDA     PACTL
        ORA     #$01        ; ENABLE BIT 0
        STA     PACTL
        RTS

   ; DISABLE PROCEED INTERRUPT

DIPRCD: LDA     PACTL
        AND     #$FE        ; DISABLE BIT0
        STA     PACTL
        RTS

   ; GET ZIOCB DEVNO - 1 INTO X

GDIDX:  LDX     ZICDNO      ; IOCB UNIT #
        DEX                 ; - 1
        RTS

    ; Convert char in A from lower-case to upper
TOUPPER:
        CMP     #'a'        ; SKip if < 'a'
        BCC     @+
        CMP     #'z'+1      ; Skip if > 'z'
        BCS     @+
        AND     #$5F        ; Disable high-bit and convert to upper
 @:     RTS

;---------------------------------------
; Proceed Vector
;---------------------------------------

PRCVEC: LDA     #$01
        STA     TRIP
        PLA
        RTI

; End Proceed Vector
;---------------------------------------

;---------------------------------------
; Reset LNBUF
;---------------------------------------
; Normally this routine is at $DA51
; But some programs will bank-switch
; that portion of ROM to RAM
;---------------------------------------

LDBUFA: LDA     #$05
        STA     INBUFF+1
        LDA     #$82        ; Normally $80. 2 for headroom
        STA     INBUFF
        RTS

; End Reset LNBUF
;---------------------------------------

;---------------------------------------
; Skip spaces
;---------------------------------------
; Normally this routine is at $DBA1
; But some programs will bank-switch
; that portion of ROM to RAM
;---------------------------------------

SKPSPC: LDY     CIX
        LDA     #$20
@:      CMP     (INBUFF),Y
        BNE     @+
        INY
        BNE     @-
@:      STY     CIX
        RTS

; End SKPSPC
;---------------------------------------


;---------------------------------------
; Return next free IOCB channel
; On return X contains IOCB channel
; such as $10, $20, .. $70
; On return:
; - Carry clear -> success
; - Carry set -> failure
;---------------------------------------
NEXT_IOCB:
        CLC                 ; Default to success
        LDX     #$10
NEXT_IOCB_LOOP:
        LDA     IOCB,X
        BMI     NEXT_IOCB_DONE  ; Found $FF = Free channel
        TXA
        ADC     #$10            ; Point to the next channel
        BMI     NEXT_IOCB_ERROR ; If $80, no more and it sets N flag
        TAX
        BNE     NEXT_IOCB_LOOP  ; Try next channel

NEXT_IOCB_ERROR:
        LDA     #<NEXT_IOCB_ERROR_STR
        LDY     #>NEXT_IOCB_ERROR_STR
        JSR     PRINT_STRING
        SEC

NEXT_IOCB_DONE:
        RTS

NEXT_IOCB_ERROR_STR:
       .BYTE    'TOO MANY FILES OPEN',EOL

; End NEXT_IOCB
;---------------------------------------


;---------------------------------------
; Print EOL-terminated string
; A: String Buffer Lo
; Y: String Buffer Hi
;---------------------------------------
PRINT_STRING:

        LDX     #$00
    ;-----------------------------------
    ; String Buffer
    ;-----------------------------------
        STA     ICBAL,X
        TYA
        STA     ICBAH,X

        LDA     #PUTREC
        STA     ICCOM,X

    ;-----------------------------------
    ; String Length
    ;-----------------------------------
        LDA     #$80
        STA     ICBLL,X
        LDA     #$00
        STA     ICBLH,X

    ;-----------------------------------
    ; Call to CIO
    ;-----------------------------------
        LDA     #PUTREC
        STA     ICCOM,X
        JMP     CIOV

;---------------------------------------
; On a device ERROR (144), STATUS-poll the failed op's unit (still
; in DUNIT) and return its extended code from DVSTAT+3 in Y.  Other
; codes (139 NAK = bad command, bus errors) pass through unchanged.
;---------------------------------------
GETEXT:
        CPY     #144
        BNE     GETEXT2
        LDA     DUNIT
        STA     STADCB+1
        LDA     #<STADCB
        LDY     #>STADCB
        JSR     DOSIOV
        LDY     DVSTAT+3
GETEXT2:
        RTS

;---------------------------------------
; Print integer error number from DOSIOV
; Y: Return code from DOSIOV
;---------------------------------------
PRINT_ERROR:
        CPY     #$01        ; Exit if success (1)
        BEQ     PRINT_ERROR_DONE

    ; On a device ERROR (144), fetch the extended code.
        JSR     GETEXT

PRINT_ERROR_NEXT:
    ;-----------------------------------
    ; Convert error code to ASCII
    ;-----------------------------------
    ; Call subroutines in ROM to convert error from int to ascii
        STY     FR0
        LDA     #$00
        STA     FR0+1
        JSR     IFP         ; Convert error from int to floating point
        JSR     FASC        ; Convert floating point to ASCII

    ;-----------------------------------
    ; Find last char in ASCII error (noted by high bit)
    ; Unset high bit & append EOL
    ;-----------------------------------
        LDY     #$FF        ; Init counter = 0
@       INY
        LDA     (INBUFF),Y
        CMP     #$80        ; Loop until high bit is set
```

```
        BCC     @-

        AND     #$7F        ; Clear high bit
        STA     (INBUFF),Y
        INY
        LDA     #EOL        ; Append EOL
        STA     (INBUFF),Y

        LDA     INBUFF
        LDY     INBUFF+1
        JMP     PRINT_STRING

PRINT_ERROR_DONE:
        RTS

; End PRINTSCR
;---------------------------------------

ASCII2ADDR:
    ;---------------------------------------
    ; Convert 4-char ASCII string found in LNBUF
    ; to bytes found at INBUFF.
    ; Ex: "0F1A" --> $1A, $0F
    ;
    ; Input:
    ; LNBUF contains 4-char ASCII string
    ; Y contains offset from LNBUF to start
    ; of ASCII string
    ;
    ; Output:
    ; INBUFF contains 2 bytes
    ;---------------------------------------

    ;---------------------------------------
    ; ASCII hex char to integer conversion
    ; algorithm borrowed from Apple II Monitor
    ;---------------------------------------
        LDA     #$00        ; Initialize output to zero
        STA     INBUFF      ; L
        STA     INBUFF+1    ; H
NEXTHEX:
        LDA     LNBUF,Y     ; Get character for hex test.
        EOR     #%00110000  ; Maps digits to $0-$9
        CMP     #$0A        ; Digit?
        BCC     DIG         ; Yes.
        ADC     #$88        ; Map letter "A"-"F" to $FA-FF.
        CMP     #$FA        ; Hex letter?
        BCC     NOTHEX      ; No, character not hex.

DIG:    ASL
        ASL
        ASL
        ASL
        LDX     #$04        ; Shift count.

HEXSHIFT:
        ASL
        ROL     INBUFF      ; Rotate into LSD.
        ROL     INBUFF+1    ; Rotate into MSD's.
        DEX                 ; Done 4 shifts?
        BNE     HEXSHIFT    ; No, loop.
        INY                 ; Advance text index
        CPY     RBUF        ; Processed 4 characters?
        BNE     NEXTHEX     ; No, get next character.

        CLC                 ;
        RTS                 ; INBUFF contains bytes

NOTHEX:
        LDA     #<RUN_ERROR_STR
        LDY     #>RUN_ERROR_STR
        JSR     PRINT_STRING
        SEC
        RTS

RUN_ERROR_STR:
        .BYTE   'ADDR? 0000..FFFF',EOL

;---------------------------------------
PARSE_COMMAS:
;---------------------------------------
    ; On entry,
    ; LNBUF contains command line such as:
    ; SAVE.N1:ABC,1234,2345,,6789
    ; CMDSEP contain offset to beginning of comma-delimited arg
    ;
    ; In the example above CMDSEP = 5 (start of 'N1:ABC')
    ;
    ; On exit, CMDSEP[0..n] contains a sequence of
    ; offsets to each of the comma-delimited
    ; args.  And COMMA_ARGS_BITFIELD has for each bit:
    ; bit=1 if the comma-delimited arg was non-null
    ; bit=0 if the comma-delimited arg was null.
    ;
    ; In the example above,
    ; CMPSEP[0..n] = {5,12,17,22,23}
    ; where 5 -> N1:ABC
    ;       12 -> 1234
    ;       17 -> 2345
    ;       22 -> null
    ;       23 -> 6789
    ;
    ; COMMA_ARGS_BITFIELD =
    ; bit#: 7 6 5 4  3 2 1 0
    ;       0 0 0 1  0 1 1 1
    ; where bit 0 -> arg 1 (N1:ABC -> 1)
    ;       bit 1 -> arg 2 (1234   -> 1)
    ;       bit 2 -> arg 3 (2345   -> 1)
    ;       bit 3 -> arg 4 (null   -> 0)
    ;       bit 4 -> arg 5 (6789   -> 1)
    ;---------------------------------------

    ;---------------------------------------------
    ; Initialize bit field. Each comma-delimited
    ; arg successfully provided will set a bit to 1
    ;---------------------------------------------
        LDA     #%00000001
        STA     COMMA_ARGS_BITFIELD

    ; Replace commas with EOL
    ; Keep offsets to spaces in CMDSEP
        LDX     #$01                ; X is arg counter, incrs with each comma
;       STX     COMMA_ARGS_BITFIELD ; Set bit field for 1st arg
        LDY     CMDSEP

PARSE_COMMAS_LOOP:
        LDA     LNBUF,Y             ; Get char from cmd line
        CMP     #EOL                ; Quit if reached end of cmd line
        BEQ     PARSE_COMMAS_DONE
        CMP     #','                ; Is curr char a comma?
        BNE     PARSE_COMMAS_NEXT2  ; Not a comma, skip next

    ; Peek at next char. If next char is comma or EOL
    ; then this arg is null
        LDA     LNBUF+1,Y           ; Here if comma, peek at next char
        CMP     #','                ; Skip ahead if peek at next char = comma
        BEQ     PARSE_COMMAS_NEXT1  ;
        CMP     #EOL                ; Skip ahead if peek at next char = EOL
        BEQ     PARSE_COMMAS_NEXT1  ;

    ; if char is comma and arg is non-null
    ; then set flag in bitfield at bit #Y
        TXA                         ; About to clobber X, stash it
        PHA

        INX
        LDA     #%00000000          ; Initialize
        SEC                         ; Inject a 1

@:      ROL                         ; Move flag into position
        DEX
        BNE     @-

        ORA     COMMA_ARGS_BITFIELD ; Set flag in bitfield
        STA     COMMA_ARGS_BITFIELD ;

        PLA                         ; Restore X
        TAX

PARSE_COMMAS_NEXT1:
        LDA     #EOL                ; Here if comma
        STA     LNBUF,Y             ; Replace with EOL
        TYA
        CLC
        ADC     #$01
        STA     CMDSEP,X            ; Store offset to delimiter
        INX                         ; Advance CMDSEP index
PARSE_COMMAS_NEXT2:
        INY                         ; Advance to next char from cmdline
        BNE     PARSE_COMMAS_LOOP   ; Do next char
PARSE_COMMAS_DONE:
        RTS


;---------------------------------------
CHECK_IF_ROM:
;---------------------------------------
    ; Checks if cart space is ROM or RAM
    ;-----------------------------------
    ; On return
    ; If A000 = ROM then Y = 1
    ; If A000 = RAM then Y = 0
    ;-----------------------------------
        LDY     #$01        ; Assume ROM -> Y=1
        LDA     $A000       ; Try altering A000
        INC     $A000       ;
        CMP     $A000       ; If A <> A000 then RAM
        BEQ     @+          ; If A = A000 then ROM (Y=FF)
        DEY                 ; RAM -> Y=0
        STA     $A000       ; Restore altered RAM
@:      RTS

;#######################################
;#                                     #
;#       COMMAND PROCESSOR (CP)        #
;#                                     #
;#######################################

;---------------------------------------
; DOS Entry point (DOSVEC points here)
;---------------------------------------
DOS:
        ; Change border if BASIC (or something else) in ROM
        LDA     COLOR4
        STA     COLOR4_ORIG          ; Preserve border
        JSR     CHECK_IF_ROM
        TYA                         ; Y = 1 if ROM
        BEQ     @+                  ; Skip ahead if A000 is RAM
        LDA     #ROM_BORDER         ; Change border
        STA     COLOR4

        ; Bypass Autorun if OPTION switch held
@:      LDA     CONSOL
        CMP     #OPTION
        BEQ     CPLOOP

        ; Autorun injection
        LDA     #CMD_IDX.AUTORUN    ; Check for AUTORUN
        CMP     AUTORUN_FLG         ; True only on 1st entry (TODO Is this working?)
        BEQ     CPLOOP              ; Skip to Command Processor
        STA     AUTORUN_FLG         ; Change flag
        JSR     SUBMIT_AUTORUN      ; Attempt to execute autorun file


CPLOOP:
        JSR     MENU_LAUNCH ; load + run the menu module (not resident)
        JMP     CPLOOP      ; Keep looping

;---------------------------------------
; Main loop
;---------------------------------------
CP:
        LDA     #$FF        ; Clear command
        STA     CMD
        JSR     SHOWPROMPT
        JSR     GETCMD
AUTORUN_DO:
        JSR     PARSECMD
        BMI     CP_DONE     ; Skip DOCMD if CMD == $FF
        JSR     DOCMD
CP_DONE:
        RTS

;---------------------------------------
; Show Command Prompt (Nn:)
; Leading EOF requires special CIOV call
;---------------------------------------

;---------------------------------------
SHOWPROMPT:
;---------------------------------------

        LDA     DOSDR       ; Get the 1,2,... for N1,N2,...
        ORA     #'0'        ; Convert, say, 1 to '1'
        STA     PRMPT+2     ; Store in after EOL and N

        LDX     #$00
        LDA     #PUTCHR
        STA     ICCOM,X

        LDA     #<PRMPT
        STA     ICBAL,X
```

```
        LDA     #>PRMPT

        STA     ICBAH,X
        LDA     #4              ; Prompt length = 4
        STA     ICBLL,X
        TXA                     ; Still zero
        STA     ICBLH,X

        JMP     CIOV

;--------------------------------------
GETCMD:
;--------------------------------------
        LDX     #$00
        LDA     #GETREC
        STA     ICCOM,X
        LDA     #<LNBUF
        STA     ICBAL,X
        LDA     #>LNBUF
        STA     ICBAH,X
        LDA     #$7F
        STA     ICBLL,X
        JSR     CIOV

GETCMDTEST:
        JSR     QSTRIP      ; strip quotes, protect in-quote spaces
        LDY     #$00
        STY     CIX
        JSR     LDBUFA      ; Reset LNBUF to $0580
        JSR     SKPSPC      ; Advance CIX to next space

    ;---------------------------------------
    ; CMDSEP is an sequence of bytes contains
    ; indexes to chars following spaces
    ; Iterate to clear CMDSEP bytes
    ;---------------------------------------
        TYA                 ; A = 0
;NOTE DO_SAVE Experiment
;       LDX     #$02        ; for X = 2 to 0 step -1
        LDX     #$04        ; for X = 2 to 0 step -1
GETLOOP:
        STA     CMDSEP,X
        DEX
        BPL     GETLOOP     ; next X

    ; Initialize Delimiter to space
        LDA     #' '
        STA     DELIM

    ;---------------------------------------
    ; Loop until EOL is encountered
    ;---------------------------------------
        INX                 ; Reset X to 0
GETCMD_LOOP:
        LDA     (INBUFF),Y
        CMP     #EOL        ; Found EOL?
        BEQ     GETCMD_DONE ; yes. skip
        CMP     DELIM       ; Found space?
        BEQ     GETCMD_REPL ; yes. Replace it
        INY
        BNE     GETCMD_LOOP ; "always" non-zero

    ;---------------------------------------
    ; March through the cmd line and note
    ; the positions of any args as delimited
    ; by spaces or quotes. positions saved
    ; in CMDSEP,X
    ;---------------------------------------
GETCMD_REPL:
        LDA     #EOL
        STA     (INBUFF),Y
        INY
        LDA     (INBUFF),Y  ; Skip any run of spaces
        CMP     #' '
        BEQ     GETCMD_REPL ; Keep skipping spaces

    ; Here if any run of spaces has ended
    ; Are we standing on a double-quote?
        CMP     #'"'
        BNE     GETCMD_WR_OFFSET  ; No. Skip ahead

    ; Here if curr char is a double-quote
        INY                 ; Advance the command line index
        LDA     #'"'        ; Is the curr delim a double-quote?
        CMP     DELIM       ; If not, change delim to double-quote
        BNE     GETCMD_DQ_DELIM

    ; Here if curr delim is a double-quote
    ; Switch delim to space
        LDA     #' '
        STA     DELIM
        BNE     GETCMD_WR_OFFSET

    ; Here if curr delim is space
    ; Switch delim to double-quote
GETCMD_DQ_DELIM:
        STA     DELIM

    ; Note the position for the curr command-line arg
GETCMD_WR_OFFSET:
        TYA
        STA     CMDSEP,X
        INX
        CPX     #$03
        BCC     GETCMD_LOOP ; Continue searching

GETCMD_DONE:
        JSR     QRESTORE    ; turn protected $00s back into spaces
        RTS

CMDSEP: .BYTE $FF,$FF,$FF,$FF,$FF
DELIM:  .BYTE ' '

;--------------------------------------
; Quote handling for filenames with spaces.
;
; QSTRIP runs before tokenizing: it removes double-quotes from LNBUF
; in place and turns any space that was INSIDE quotes into a $00, so
; the space/comma tokenizers (GETCMDTEST, PARSE_COMMAS) don't treat it
; as an arg delimiter.  QRESTORE runs after tokenizing and turns those
; $00 sentinels back into real spaces.  This lets a name be quoted in
; any position (FROM, TO, a lone arg, ...), e.g.
;   COPY "N2:Axis Assassin.xex","N1:My File.xex"
; A $00 is safe: it never occurs in an ATASCII command line and is a
; delimiter to neither tokenizer.
;--------------------------------------
QSTRIP:
        LDX     #$00        ; dst index (compacting in place)
        LDY     #$00        ; src index
        STX     QINQ        ; in-quote flag = 0 (X is 0 here)
QST_LOOP:
        LDA     LNBUF,Y
        CMP     #EOL
        BEQ     QST_EOL
        CMP     #'"'
        BEQ     QST_TOGGLE
        CMP     #' '
        BNE     QST_STORE   ; ordinary char -> copy as-is
        LDA     QINQ        ; a space: protect it only inside quotes
        BEQ     QST_KEEPSP  ; outside quotes -> keep the space
        LDA     #$00        ; inside quotes -> $00 sentinel
        BEQ     QST_STORE   ; (always; A = 0)
QST_KEEPSP:
        LDA     #' '
QST_STORE:
        STA     LNBUF,X
        INX
        INY
        JMP     QST_LOOP
QST_TOGGLE:
        LDA     QINQ        ; flip in-quote state, drop the quote char
        EOR     #$01
        STA     QINQ
        INY
        JMP     QST_LOOP
QST_EOL:
        LDA     #EOL
        STA     LNBUF,X
        RTS

QRESTORE:
        LDX     #$00
QR_LOOP:
        LDA     LNBUF,X
        BNE     QR_NEXT     ; leave non-$00 bytes alone
        LDA     #' '
        STA     LNBUF,X
QR_NEXT:
        INX
        CPX     #$80        ; whole 128-byte LNBUF
        BCC     QR_LOOP
        RTS

;--------------------------------------
PARSECMD:
;--------------------------------------
        ;LDA     LNBUF
        LDY     #$00
        LDA     (INBUFF),Y
        CMP     #EOL        ; Quit immediately if no cmd
        BEQ     PARSECMD_DONE

        JSR     PARSE_INTRINSIC_COMMAND
        JSR     PARSE_DRIVE_CHANGE
        JSR     PARSE_EXTRINSIC_COMMAND
        JSR     PRINT_UNK_CMD
PARSECMD_DONE:
        RTS

PRINT_UNK_CMD:
        LDA     CMD
        CMP     #$FF
        BNE     PRINT_UNK_CMD_DONE
        LDA     #<UNK_CMD_ERR
        LDY     #>UNK_CMD_ERR
        JSR     PRINT_STRING
PRINT_UNK_CMD_DONE:
        RTS

UNK_CMD_ERR:
        .BYTE   'CMD?',EOL

;--------------------------------------
PARSE_INTRINSIC_COMMAND:
;--------------------------------------
        LDX     #$00        ; Initialize X for command code indexing
        LDY     #$00
        STY     CIX
        JSR     LDBUFA      ; Set INBUFF to $0580, er make that TBUF

@:      JSR     SKPSPC      ; Skip whitespace

PARSE_INTRINSIC_NEXT_CHAR:
        LDA     (INBUFF),Y
        AND     #$5F        ; Disable high-bit and convert to upper
        EOR     COMMAND,X   ; X initialized to 0 in caller
        INY
        ASL
        BEQ     PARSE_INTRINSIC_CHAR_OK

        ; Skip to next command

PARSE_INTRINSIC_NEXT_COMMAND:
        LDA     COMMAND,X
        ASL
        INX
        BCC     PARSE_INTRINSIC_NEXT_COMMAND
        LDY     CIX
        CPX     #COMMAND_SIZE

PARSE_INTRINSIC_CHAR_OK:
        INX
        BCC     PARSE_INTRINSIC_NEXT_CHAR
        STY     CIX
        LDA     (INBUFF),Y
        BMI     PARSE_INTRINSIC_RET

        JSR     SKPSPC

PARSE_INTRINSIC_RET_ERROR:
        LDX     #COMMAND_SIZE+1
PARSE_INTRINSIC_RET:
        LDA     COMMAND,X
        STA     CMD
        STA     CMDPRV
PARSE_INTRINSIC_DONE:
        RTS

; End of PARSE_INTRINSIC_COMMAND
;--------------------------------------

;--------------------------------------
PARSE_DRIVE_CHANGE:
;--------------------------------------
        LDX     #$03        ; Check for EOL in pos 3
        LDA     LNBUF,X
        CMP     #EOL
        BNE     PARSE_DRIVE_CHANGE_DONE
        DEX                 ; go back one char
        LDA     LNBUF,X
        CMP     #':'        ; Check for colon.
        BNE     PARSE_DRIVE_CHANGE_DONE
```

```
        LDA     #CMD_IDX.DRIVE_CHG
        STA     CMD
PARSE_DRIVE_CHANGE_DONE:
        RTS

;--------------------------------------
PARSE_EXTRINSIC_COMMAND:
;--------------------------------------
    ; Quit if CMD has been found earlier
        LDX     CMD         ; Undefined CMD = $FF
        INX                 ; now 0 if undefined
        BNE     PARSE_DRIVE_CHANGE_DONE ; Exit if defined CMD

    ; Here if  CMD is undefined.
    ; From here we'll assume it's a filename for a executable
    ; and attempt to LOAD it. But first append a ".COM"
    ; and shift the filename to the right

    ; Find offset to EOL
        LDY     #$FF
@       INY
        LDA     (INBUFF),Y
        CMP     #EOL
        BNE     @-

    ; Y contains offset to last char & stash it
        TYA
        PHA

    ; Advance Y to allow the 5 chars of '.COM',EOL
        CLC
        ADC     #$05
        TAY

    ; Append .COM, EOL
        LDX     #$04
@:      LDA     CMDEXT,X
        STA     (INBUFF),Y
        DEY
        DEX                     ;
        BPL     @-

    ; Shift executable name to the right to allow room for PREPEND_DRIVE
    ; Stack contains offset to last char
    ; Y is still counting down from where '.COM' was appended.
        PLA                 ; Get offset to last char of executable
        TAX                 ; It'll be used for indexing
        DEX                 ; Skip original EOF
SHIFT_LOOP:
        LDA     LNBUF,X     ; Get source ch
@:      STA     (INBUFF),Y  ; Copy ch to new location
        DEY                 ; Point to next dest ch
        DEX                 ; Point to next source ch
        BPL     SHIFT_LOOP  ; Until X = 0

    ; Let DO_LOAD attempt to execute the file
        LDA     #$01        ; Point to start of filename
        STA     CMDSEP      ; so DO_LOAD will treat it like
        PLA                 ; Strip the last return addr off the stack.
        PLA                 ; Othwerwise unknown cmd error later.
        JMP     DO_LOAD     ; 'LOAD filename'

CMDEXT:
        .BYTE   '.COM',EOL

; End of PARSE_EXTRINSIC_COMMAND
;--------------------------------------

;--------------------------------------
DOCMD:
;--------------------------------------
        LDX     CMD
        BMI     DOCMD_DONE  ; Unassigned command = $FF
        LDA     CMD_TAB_H,X ; Get hi-byte of subroutine's addr
        PHA                 ; Push it to the stack
        LDA     CMD_TAB_L,X ; Get lo-byte of subroutine's addr
        PHA                 ; Push it to the stack
DOCMD_DONE:
        RTS                 ; Use stack & RTS to jump to subroutine

; End of DOCMD
;--------------------------------------

;--------------------------------------
DO_DRIVE_CHG:
;--------------------------------------
        LDA     LNBUF
        AND     #%11011111  ; Convert lower to upper
        CMP     #'N'
        BNE     DO_DRIVE_CHG_ERROR
        STA     PRMPT+1
        LDA     LNBUF+1
        CMP     #'1'        ; Skip if < '1'
        BCC     DO_DRIVE_CHG_ERROR
        CMP     #'9'        ; Skip if >= '9'
        BCS     DO_DRIVE_CHG_ERROR
        AND     #$0F        ; Convert, say, to 1-8
        STA     DOSDR
        RTS
DO_DRIVE_CHG_ERROR:
        LDA     #<CDERR
        LDY     #>CDERR
        JMP     PRINT_STRING

; End of DOCMD

;--------------------------------------
; Returns DOSDR in X
; If arg1 contains Nn: then reg X = n
; Otherwise X = DOSDR (from curr prompt)
;--------------------------------------
GET_DOSDR:
;--------------------------------------

        JSR     LDBUFA      ; Reset INBUFF to LNBUF
        LDX     DOSDR       ; Set DUNIT(X) to curr drive

    ;--------------------------------------
    ; Consider arg1 = N2:TNFS://localhost/
    ; Check arg1 for ":" in 3rd position
    ; if found then use char in 2nd position ('2') as DOSDR
    ; First, change INBUFF to point to beg. of 1st arg
    ;--------------------------------------
        LDA     CMDSEP              ; arg offset
        BEQ     GET_DOSDR_DONE      ; first, if arg1 not pop, leave X as-is & quit

GET_DOSDR_ALT_ENTRY:
        CLC                         ; Advance pointer to LNBUF
        ADC     INBUFF
        STA     INBUFF
        BCC     GET_DOSDR_NEXT
        INC     INBUFF+1

GET_DOSDR_NEXT:
        LDY     #$02            ; Check for ':' in 3rd pos
        LDA     (INBUFF),Y
        CMP     #':'
        BNE     GET_DOSDR_DONE  ; Not found, skip & use default
        DEY
        LDA     (INBUFF),Y
        AND     #$0F            ; Convert, say, '2' to 2
        TAX                     ; Return DOSDR in X

GET_DOSDR_DONE:
        RTS

;--------------------------------------
DO_GENERIC:
;--------------------------------------

    ;--------------------------------------
    ; SIO call for NCD,{MK,RM}DIR,DEL,RENAME
    ;--------------------------------------

    ;--------------------------------------
    ; Get Fujinet SIO command (~ $21 for DEL)
    ; X = table index from caller
    ;--------------------------------------
        LDA     CMD_DCOMND,X
        STA     GENDCB+DCB_IDX.DCOMND

        CMP     #CMD_USER       ; Skip prepending devspec for SSH
        BEQ     DO_GENERIC_NEXT
        CMP     #CMD_PASS
        BEQ     DO_GENERIC_NEXT

    ;--------------------------------------
    ; Get DOSDR from either arg1 or curr drive
    ;--------------------------------------
        JSR     GET_DOSDR    ; X will contain int of n in Nn:
        STX     GENDCB+DCB_IDX.DUNIT
        JSR     PREPEND_DRIVE

    ;--------------------------------------
    ; If this is NCD ensure a '/' char is the last char
    ;--------------------------------------
        LDA     GENDCB+DCB_IDX.DCOMND
        CMP     #CMD_CD         ; Is this an NCD command?
        BNE     DO_GENERIC_NEXT ; No. skip

        LDA     CMDSEP
        BEQ     NCD_ERROR
        ;JSR     APPEND_SLASH    ; Append '/' to path if missing

DO_GENERIC_NEXT:
    ;--------------------------------------
    ; Populate the DCB
    ;--------------------------------------
        LDA     DOSDR
        STA     STADCB+DCB_IDX.DUNIT ; Yes. Status (not typo)
;        STA     GENDCB+1   ; 20221105 - commented-out. checking for bug...
        LDA     INBUFF
        STA     GENDCB+DCB_IDX.DBUFL
        LDA     INBUFF+DCB_IDX.DUNIT
        STA     GENDCB+DCB_IDX.DBUFH

    ;--------------------------------------
    ; Send the command to FujiNet
    ;--------------------------------------
        LDA     #<GENDCB
        LDY     #>GENDCB
        JSR     DOSIOV
        JSR     PRINT_ERROR

GEN_UNMOUNT:
    ;--------------------------------------
    ; if DEL or RENAME, then remount drive
    ;--------------------------------------
        LDA     CMDPRV
        CMP     #CMD_IDX.DEL
        BEQ     GEN_REMOUNT
        CMP     #CMD_IDX.RENAME
        BNE     GENDONE
GEN_REMOUNT:
        JMP     REMOUNT_DRIVE

GENDONE:
        RTS

;--------------------------------------
NCD_ERROR:
;--------------------------------------
        LDA     #<NCD_ERROR_STR
        LDY     #>NCD_ERROR_STR
        JSR     PRINT_STRING
        LDY     #$01        ; Return error
        RTS
    ;--------------------------------------
    ; Close
    ;--------------------------------------
        LDX     #$10        ; File #1
        LDA     #$0C        ; Close #1 first
        STA     ICCOM,X
        JSR     CIOV

NCD_ERROR_STR:
        .BYTE   'PATH?',EOL

;--------------------------------------
GENDCB:
        .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   $FF         ; DCOMND
        .BYTE   $80         ; DSTATS
        .BYTE   $FF         ; DBUFL
        .BYTE   $FF         ; DBUFH
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $00         ; DBYTL
        .BYTE   $01         ; DBYTH
        .BYTE   $00         ; DAUX1
        .BYTE   $00         ; DAUX2

; End of DO_GENERIC
;--------------------------------------

;--------------------------------------
DO_LOAD:
;--------------------------------------
    ; Binary Loader based on work by Richard J. Kalagher 07-AUG-1983

    ; Open file
```

```
        LDA     CMDSEP          ; Quit if no arg1
        BNE     LOAD_NEXT1
        JMP     LOAD_ERROR

LOAD_NEXT1:
    ; Point INBUFF to start of filename
        CLC
        ADC     INBUFF          ; [A] contains offset to arg1
        STA     INBUFF
        BCC     LOAD_NEXT2
        INC     INBUFF+1

LOAD_NEXT2:
        JSR     LOAD_NTRANS     ; Disable any EOL translation
        JSR     LOAD_SETUP      ; Set up run and init to RTS
        LDA     #OINPUT         ; A arg needed in LOAD_OPEN
        JSR     LOAD_OPEN       ; Open the file
        CPY     #$01            ; Quit if unable to open
        BNE     R

        LDA     #$FF
        STA     BIN_1ST
        JSR     LOAD_READ2
        JSR     LOAD_CHKFF
        CPY     #$01
        BNE     R

        INC     BIN_1ST
    ; Process each payload
GETFIL: JSR     LOAD_READ2      ; Get two bytes (binary header)
        BMI     R               ; Exit if EOF hit
        JSR     LOAD_INIT       ; Set init default
        LDX     #$01
        JSR     LOAD_CHKFF      ; Check if header (and start addr, too)
        JSR     LOAD_STRAD      ; Put start address in
        JSR     LOAD_READ2      ; Get to more butes (end addr)
        JSR     LOAD_ENDAD      ; Put end address in
        JSR     LOAD_BUFLEN     ; Calculate buffer length
        JSR     LOAD_GETDAT     ; Get the data record
        BPL     @+              ; Was EOF detected?
        JSR     JSTART          ; Yes. Go to RUNAD
@:      JSR     JINIT           ; Attempt initialization
        JMP     GETFIL          ; Process next payload

JINIT:  JMP     (INITAD)        ; Will either RTS or perform INIT
JSTART: JMP     (RUNAD)         ; Godspeed.
R:      RTS                     ; Stunt-double for (INITAD),(RUNAD)

;---------------------------------------
LOAD_SETUP:
;---------------------------------------
        LDA     #<R
        STA     RUNAD
        LDA     #>R
        STA     RUNAD+1
        RTS

;---------------------------------------
LOAD_INIT:
;---------------------------------------
        LDA     #<R
        STA     INITAD
        LDA     #>R
        STA     INITAD+1
        RTS

;---------------------------------------
LOAD_OPEN:
;---------------------------------------
        PHA                     ; Save data direction passed in A
        JSR     GET_DOSDR       ; Get DUNIT
        STX     OPNDCB+DCB_IDX.DUNIT
        STX     GETDCB+DCB_IDX.DUNIT ; Set DUNIT FOR READ
        JSR     PREPEND_DRIVE

        LDA     INBUFF          ; Register location of filename
        STA     OPNDCB+DCB_IDX.DBUFL
        LDA     INBUFF+DCB_IDX.DUNIT
        STA     OPNDCB+DCB_IDX.DBUFH

        PLA                     ; A = data direction (4=in, 8=out)
        STA     OPNDCB+DCB_IDX.DAUX1
        LDA     #$00            ; AUX2: No translation
        STA     OPNDCB+DCB_IDX.DAUX2

        LDA     #<OPNDCB
        LDY     #>OPNDCB
        JSR     DOSIOV

        PHA
        JSR     PRINT_ERROR
        PLA
        TAY

        RTS

;---------------------------------------
LOAD_NTRANS:
;---------------------------------------
    ; Disable any EOL transation otherwise
    ; binary data will be corrupted during load
    ;-----------------------------------
        JSR     GET_DOSDR       ; Get DUNIT
        STX     NTRDCB+DCB_IDX.DUNIT
        LDA     #$00            ; Translation mode (0 = NONE)
        STA     NTRDCB+DCB_IDX.DAUX2
        LDA     #<NTRDCB
        LDY     #>NTRDCB
        JSR     DOSIOV
        JMP     PRINT_ERROR

;---------------------------------------
LOAD_READ2:
;---------------------------------------

    ;-----------------------------------
    ; This is accomplished by abusing the LOAD_GETDAT
    ; routine by stuffing the buffer addr (BAL/H)
    ; into the payload Start/End addrs. We're doing
    ; this in case a payload  header straddles a
    ; cache boundary. LOAD_GETDAT has the logic for
    ; dealing with that.
    ;-----------------------------------
        LDA     #<BAL
        STA     STL         ; Payload start address
        LDA     #>BAL
        STA     STH

        LDA     #<BAH
        STA     ENL         ; Payload end address
        LDA     #>BAH
        STA     ENH

        LDX     #$02
        STX     BLL         ; Payload size (2)
        LDA     #$00
        STA     BLH

        JMP     LOAD_GETDAT ; Read 2 bytes

;---------------------------------------
LOAD_CHKFF:
;---------------------------------------
    ; On 1st pass, check for binary signature (FF FF)
    ; On 2..n passes, Skip FF FF (if found)
    ; and read next 2 bytes
    ;-----------------------------------

        LDA     #$FF
        CMP     BAL         ; Is 1st byte FF?
        BNE     NOTFF       ; If no, skip down.
        CMP     BAH         ; Is 2nd byte FF?
        BNE     NOTFF       ; If no, skip down.

    ;-----------------------------------
    ; Here if FF FF tags found.
    ; On 1st pass, we're done.
    ; On 2..n passes, read next 2 bytes and leave.
    ;-----------------------------------
        CMP     BIN_1ST     ; Is this 1st pass?
        BEQ     NOTFF_DONE  ; If yes, then we're done here.
        JMP     LOAD_READ2  ;

    ;-----------------------------------
    ; Here if FF FF tags NOT found.
    ; On 1st pass, print error.
    ; On 2..n passes, the 2 bytes = payload start addr.
    ;-----------------------------------
NOTFF:  LDY     #$01        ; Preload success return code
        CMP     BIN_1ST     ; A still has FF. BIN_1ST = FF on first pass
        BNE     NOTFF_DONE  ; Not 1st pass, exit with success.

NOTFF_ERR:
        LDA     #<LOAD_ERROR_STR2
        LDY     #>LOAD_ERROR_STR2
        JSR     PRINT_STRING

        LDY     #$FF        ; Return failure
NOTFF_DONE:
        RTS

LOAD_ERROR_STR2:
        .BYTE   'NOT A BINARY FILE',EOL

;---------------------------------------
LOAD_STRAD:
;---------------------------------------
    ; Save payload start address into STL2/STLH2.
    ; Otherwise it will get clobbered
    ; when reading payload end address.
        LDA     RBUF
        STA     STL2
        LDA     RBUF+1
        STA     STH2
        RTS

;---------------------------------------
LOAD_ENDAD:
;---------------------------------------
    ; Save payload end address
        LDA     STL2
        STA     STL
        LDA     STH2
        STA     STH

        LDA     RBUF
        STA     ENL
        LDA     RBUF+1
        STA     ENH
        RTS

;---------------------------------------
LOAD_BUFLEN:
;---------------------------------------
    ; Calculate buffer length (end-start+1)

    ; Calc buffer size Lo
        LDA     ENL
        SEC
        SBC     STL
        STA     BLL     ; Buffer Length Lo

    ; Calc buffer size Hi
        LDA     ENH     ; Calc buffer size Hi
        SBC     STH
        STA     BLH     ; Buffer Length Hi

    ; Add 1
        CLC
        LDA     BLL
        ADC     #$01
        STA     BLL

        LDA     BLH
        ADC     #$00    ; Take care of any carry
        STA     BLH

        RTS


.IFDEF BURST_MODE
.ECHO "FOO!"
;---------------------------------------
LOAD_GETDAT:
;---------------------------------------
    ;-----------------------------------
    ; Fill out the DCB
    ;-----------------------------------
        JSR     GET_DOSDR
        STX     BINDCB+1        ; DUNIT

        LDA     STL
        STA     BINDCB+4        ; DBUFL
        LDA     STH
        STA     BINDCB+5        ; DBUFH
        LDA     BLL
        STA     BINDCB+8        ; DBYTL
        STA     BINDCB+10
        LDA     BLH
        STA     BINDCB+9        ; DBYTH
        STA     BINDCB+11
```

```
    ;--------------------------------------
    ; Send Read request to SIO
    ;--------------------------------------
        LDA     #<BINDCB
        LDY     #>BINDCB
        JSR     DOSIOV
        JSR     PRINT_ERROR     ; Show any errors

    ;--------------------------------------
    ; Get status (updates DVSTAT, DSTATS)
    ;--------------------------------------
        LDA     BINDCB+1
        STA     STADCB+1
        LDA     #<STADCB
        LDY     #>STADCB
        JSR     DOSIOV

    ; Check if EOF (current requested chunk completed?)
        LDA     #EOF
        CMP     DVSTAT+3
        BEQ     LOAD_GETDAT_DONE
        JMP     PRINT_ERROR

LOAD_GETDAT_DONE:
    ; Check if 0 bytes remaining
        LDA     DVSTAT
        BNE     LOAD_GETDAT_DONE2
        LDA     DVSTAT+1
        BNE     LOAD_GETDAT_DONE2
        LDY     #$FF
        RTS

LOAD_GETDAT_DONE2:
        LDY     #$01            ; Return success
        RTS

BINDCB:
       .BYTE    DEVIDN      ; DDEVIC
       .BYTE    $FF         ; DUNIT
       .BYTE    'R'         ; DCOMND
       .BYTE    $40         ; DSTATS
       .BYTE    $FF         ; DBUFL
       .BYTE    $FF         ; DBUFH
       .BYTE    $FE         ; DTIMLO
       .BYTE    $00         ; DRESVD
       .BYTE    $FF         ; DBYTL
       .BYTE    $FF         ; DBYTH
       .BYTE    $FF         ; DAUX1
       .BYTE    $FF         ; DAUX2

.ELSE
.ECHO "BAR"
;--------------------------------------
LOAD_GETDAT:
;--------------------------------------
    ; Definitions:
    ; HEAD = requested bytes that will be found in current cache (< 512 bytes)
    ; BODY = contiguous 512 byte sections. BODY = n * 512 bytes)
    ; TAIL = any bytes remaining after BODY (< 512 bytes)

        JSR     GET_DOSDR
        STX     BINDCB+DCB_IDX.DUNIT

        JSR     GETDAT_CHECK_EOF    ; Check EOF before proceeding
        BPL     GETDAT_NEXT1        ; If true, then EOF found. Exit
        RTS

    ; Check if bytes requested BL < DVSTAT (bytes waiting in cache)
GETDAT_NEXT1:
        LDA     DVSTAT
        CMP     BLL
        LDA     DVSTAT+1
        SBC     BLH
        BCS     GETDAT_OPT2     ; BL <= BW (bytes waiting)

GETDAT_OPT1:
    ;---------------------------------
    ; Here if bytes requested > bytes
    ; remaining in cache
    ;---------------------------------

    ;---------------------------------
    ; Head = BW (bytes waiting)
    ;---------------------------------
        LDA     DVSTAT
        STA     HEADL
        LDA     DVSTAT+1
        STA     HEADH

    ;---------------------------------
    ; Tail = (BL - HEAD) mod 512
    ;---------------------------------
        SEC
        LDA     BLL
        SBC     HEADL
        AND     #$FF
        STA     TAILL
        LDA     BLH
        SBC     HEADH
        AND     #$01
        STA     TAILH

    ;----------------------------------
    ; Body = BL - HEAD - TAIL
    ;----------------------------------
        ; 1. Body = BL - HEAD
        ;--------------------------------
        SEC
        LDA     BLL
        SBC     HEADL
        STA     BODYL
        LDA     BLH
        SBC     HEADH
        STA     BODYH

        ;--------------------------------
        ; 2. Body = Body - HEAD
        ;--------------------------------
        SEC
        LDA     BODYL
        SBC     TAILL
        STA     BODYL
        LDA     BODYH
        SBC     TAILH
        STA     BODYH

        JMP     GETDAT_READ

GETDAT_OPT2:
    ;---------------------------------
    ; Here if bytes requested <= bytes
    ; remaining in cache
    ;---------------------------------
    ; Head = BL, TAIL = BODY = 0
    ;---------------------------------
        LDA     BLL
        STA     HEADL
        LDA     BLH
        STA     HEADH
        LDA     #$00
        STA     TAILL
        STA     TAILH
        STA     BODYL
        STA     BODYH

;--------------------------------------
GETDAT_READ:
;--------------------------------------
    ;--------------------------------------
    ; Read HEAD bytes
    ;--------------------------------------
        LDA     HEADL
        STA     BLL
        LDA     HEADH
        STA     BLH
        JSR     GETDAT_DOSIOV
        BPL     GETDAT_BODY ; Skip ahead if no problems
        RTS                 ; Bail if error

    ;--------------------------------------
    ; Read BODY bytes
    ;--------------------------------------
GETDAT_BODY:
        LDX     BODYH
GETDAT_BODY_LOOP:
        BEQ     GETDAT_TAIL ; Skip if less than a page to read

        LDA     #$00
        STA     BLL         ; Buffer length
        LDA     #$02        ; 512 bytes at a time
        STA     BLH

        TXA                 ; Stash our loop index (X)
        PHA                 ; onto the stack
        JSR     GETDAT_DOSIOV
        BPL     @+          ; Skip ahead if no problems
        PLA                 ; Here if problem. Clean up stack
        TYA                 ; Reset N status flag before returning
        RTS                 ; Bail if error

@:      PLA                 ; Retrieve our loop index
        TAX                 ; and xfer it back into X
        DEX                 ; -2 (we pull 0200 bytes at a time)
        DEX                 ;
        BNE     GETDAT_BODY_LOOP

GETDAT_TAIL:
    ;--------------------------------------
    ; Read TAIL bytes
    ;--------------------------------------
        LDA     TAILL
        STA     BLL
        LDA     TAILH
        STA     BLH

;--------------------------------------
GETDAT_DOSIOV:
;--------------------------------------
    ; Bail if BL = 0
        LDA     BLL
        BNE     @+
        LDA     BLH
        BEQ     CHECK_EOF_DONE

@:
    ; SIO READ
        LDA     STL
        STA     BINDCB+DCB_IDX.DBUFL    ; Start Address Lo
        LDA     STH
        STA     BINDCB+DCB_IDX.DBUFH    ; Start Address Hi
        LDA     BLL
        STA     BINDCB+DCB_IDX.DBYTL    ; Buffer Size Lo
        STA     BINDCB+DCB_IDX.DAUX1
        LDA     BLH
        STA     BINDCB+DCB_IDX.DBYTH    ; Buffer Size Hi
        STA     BINDCB+DCB_IDX.DAUX2

    ;--------------------------------------
    ; Send Read request to SIO
    ;--------------------------------------
        LDA     #<BINDCB
        LDY     #>BINDCB
        JSR     DOSIOV
        JSR     PRINT_ERROR

    ;--------------------------------------
    ; Advance start address by buffer length
    ;--------------------------------------
        CLC
        LDA     STL
        ADC     BLL
        STA     STL

        LDA     STH
        ADC     BLH
        STA     STH

GETDAT_CHECK_EOF:
    ; Get status (updates DVSTAT, DSTATS)
        LDA     BINDCB+DCB_IDX.DUNIT
        STA     STADCB+DCB_IDX.DUNIT
        LDA     #<STADCB
        LDY     #>STADCB
        JSR     DOSIOV

    ; Return -1 if DVSTAT == $0000 and DVSTAT+3 == EOF
        LDA     DVSTAT
        BNE     CHECK_EOF_DONE

        LDA     DVSTAT+1
        BNE     CHECK_EOF_DONE

        LDA     #EOF
        CMP     DVSTAT+3        ; Is it EOF
        BNE     CHECK_EOF_DONE  ; No? Go to success
        LDY     #$FF            ; Yes? Return -1
        RTS

CHECK_EOF_DONE:
        LDY     #$01        ; Return success
        RTS
```

```
BINDCB:
       .BYTE    DEVIDN      ; DDEVIC
       .BYTE    $FF         ; DUNIT
       .BYTE    'R'         ; DCOMND
       .BYTE    $40         ; DSTATS
       .BYTE    $FF         ; DBUFL
       .BYTE    $FF         ; DBUFH
       .BYTE    $FE         ; DTIMLO
       .BYTE    $00         ; DRESVD
       .BYTE    $FF         ; DBYTL
       .BYTE    $FF         ; DBYTH
       .BYTE    $FF         ; DAUX1
       .BYTE    $FF         ; DAUX2
.ENDIF

;---------------------------------------
LOAD_CLOSE:
;---------------------------------------
        LDA     BINDCB+DCB_IDX.DUNIT
        STA     CLODCB+DCB_IDX.DUNIT
        LDA     #<CLODCB
        LDY     #>CLODCB
        JMP     DOSIOV

;---------------------------------------
LOAD_ERROR:
;---------------------------------------
        LDA     #<MISSING_FILE_STR
        LDY     #>MISSING_FILE_STR
        JMP     PRINT_STRING

;---------------------------------------
DO_AUTORUN:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.AUTORUN
        JMP     DO_OVERLAY

;---------------------------------------
DO_BASIC:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.BASIC
        BNE     JMP_OVERLAY

;---------------------------------------
DO_DIR:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.DIR
        BNE     JMP_OVERLAY

;---------------------------------------
DO_DUMP:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.DUMP
        BNE     JMP_OVERLAY

;---------------------------------------
DO_FILL:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.FILL
        BNE     JMP_OVERLAY

;---------------------------------------
DO_HELP:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.HELP
        BNE     JMP_OVERLAY

;---------------------------------------
DO_NCOPY:
;---------------------------------------
        LDA     #$FF        ; Force DO_OVERLAY to always re-read
        STA     OVLPRV      ; code from ATR sector (cache confusion)
        LDX     #OVL_IDX.NCOPY
        BNE     JMP_OVERLAY

;---------------------------------------
DO_NCOPY1:
;---------------------------------------
        LDA     #$FF        ; Force DO_OVERLAY to always re-read
        STA     OVLPRV      ; code from ATR sector (cache confusion)
        LDX     #OVL_IDX.NCOPY1
        BNE     JMP_OVERLAY

;---------------------------------------
DO_NCOPY2:
;---------------------------------------
        LDX     #OVL_IDX.NCOPY2
        BNE     JMP_OVERLAY

;---------------------------------------
DO_NDEL:
;---------------------------------------
    ; DEL/ERA/ERASE: load the scan overlay, which decides between a
    ; plain single-file delete and the wildcard (Y/N) delete driver.
        LDX     #OVL_IDX.NDEL
        BNE     JMP_OVERLAY

;---------------------------------------
DO_NTRANS:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.NTRANS
        BNE     JMP_OVERLAY

;---------------------------------------
DO_REENTER:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.REENTER
        BNE     JMP_OVERLAY

;---------------------------------------
DO_SAVE:
;---------------------------------------
    ; Load sector from NOS ATR into RAM and jump to it.
        LDX     #OVL_IDX.SAVE
        BNE     JMP_OVERLAY

;---------------------------------------
DO_XEP:
;---------------------------------------
        LDX     #OVL_IDX.XEP
JMP_OVERLAY:
        JMP     DO_OVERLAY
```

```
;---------------------------------------
DO_OVERLAY:
;---------------------------------------
    ; Use table to get sector where overlay is stored
        LDA     OVL_SECT_TAB_L,X
        STA     GET_SECTOR_DCB+DCB_IDX.DAUX1

    ; Currently, all overlays exist in sectors < 01xx,
    ; so $00 can be assumed for high byte of sector number.
;       LDA     OVL_SECT_TAB_H,X
        LDA     #$00
        STA     GET_SECTOR_DCB+DCB_IDX.DAUX2

    ; Get the number of sectors to load
        LDA     OVL_SECT_CNT_TAB,X
        STA     SECT_CNT

    ; Quit if requested overlay command is already in memory
        LDA     CMD             ; Get current command
        CMP     OVLPRV          ; Is this already in memory?
        BEQ     OVERLAY_DONE    ; Quit if already in memory
        STA     OVLPRV          ; Update previous overlay command

    ; Initialize the base address for the code to be loaded
        LDA     #<OVLBUF
        STA     GET_SECTOR_DCB+DCB_IDX.DBUFL
        LDA     #>OVLBUF
        STA     GET_SECTOR_DCB+DCB_IDX.DBUFH

    ; Get number of sectors to read
OVERLAY_LOOP:
        LDA     #<GET_SECTOR_DCB
        LDY     #>GET_SECTOR_DCB
        JSR     DOSIOV
        DEC     SECT_CNT
        BEQ     OVERLAY_DONE
    ; Increment the load sector by 1
        CLC
        INC     GET_SECTOR_DCB+DCB_IDX.DAUX1
        BCC     @+
        INC     GET_SECTOR_DCB+DCB_IDX.DAUX2
    ; Increment load address by 1 sector distance ($80)
@:      CLC
        LDA     GET_SECTOR_DCB+DCB_IDX.DBUFL
        ADC     #SECTOR_SIZE
        STA     GET_SECTOR_DCB+DCB_IDX.DBUFL
        BCC     @+
        INC     GET_SECTOR_DCB+DCB_IDX.DBUFH
@:      JMP     OVERLAY_LOOP
OVERLAY_DONE:
        JMP     OVLBUF

GET_SECTOR_DCB:
       .BYTE    $31         ; DDEVIC - Floppy
       .BYTE    $01         ; DUNIT
       .BYTE    'R'         ; DCOMND
       .BYTE    $40         ; DSTATS
       .BYTE    <OVLBUF     ; DBUFL - Destination
       .BYTE    >OVLBUF     ; DBUFH
       .BYTE    $FE         ; DTIMLO
       .BYTE    $00         ; DRESVD
       .BYTE    $80         ; DBYTL - Bytes to read
       .BYTE    $00         ; DBYTH
       .BYTE    $FF         ; DAUX1 - Sector
       .BYTE    $FF         ; DAUX2

;#####################################
;#   WILDCARD COPY  (resident glue)     #
;#####################################
; The wildcard-copy DRIVER is a load-on-demand module (WILD_MOD, near
; end of file) so it costs almost no resident RAM -- only this loader
; and the scratch equates live here.  OVL_NCOPY scans FROM for '*'/'?'
; inline and JMPs to WILD_LAUNCH.  Scratch sits above the per-file NCOPY
; burst buffer (MEMLO..MEMLO+$2000) and below the module, so it survives
; each copy.  Indirect access uses ZP INBUFF; WFROM/WNPTR are holding vars.
WNBUF   =   $4000
WPREFIX =   WNBUF               ; FROM dir prefix (up to last '/' or ':'), EOL-term
WTO     =   WNBUF+$80           ; TO string, EOL-term
WNAMES  =   WNBUF+$100          ; matched names, each EOL-term; list ends with $00
WILD_RUN =  $4300               ; module run address (above WNAMES + burst buffer)

; WILD_LAUNCH: load the wildcard module to WILD_RUN and run it.  Does NOT
; reset the stack: the module RTSes to the command that invoked the copy.
WILD_LAUNCH:
        LDA     #WILD_SECT
        STA     GET_SECTOR_DCB+DCB_IDX.DAUX1
        LDA     #WILD_CNT
        STA     SECT_CNT
        LDA     #<WILD_RUN
        STA     GET_SECTOR_DCB+DCB_IDX.DBUFL
        LDA     #>WILD_RUN
        STA     GET_SECTOR_DCB+DCB_IDX.DBUFH
    ; DAUX2 (sector hi) is already 0 here: DO_OVERLAY zeroed it loading
    ; OVL_NCOPY, and all these sectors are < 256.  Reuse the menu
    ; module's sector-read loop, then run the module (no stack reset --
    ; the module RTSes back to the command that started the copy).
        JSR     MENU_LOAD_LOOP
        JMP     WILD_RUN


;---------------------------------------
DO_NPWD:
;---------------------------------------
        LDA     #EOL        ; Truncate buffer
        STA     RBUF

        JSR     GET_DOSDR   ; X will contain n in Nn:
NPWD_ENTRY:
        STX     PWDDCB+DCB_IDX.DUNIT

        LDA     #<PWDDCB
        LDY     #>PWDDCB
        JSR     DOSIOV
        JSR     PRINT_ERROR

    ;-------------------------------------
    ; If we entered DO_NPWD from REMOUNT_DRIVE
    ; then skip printing output
    ;-------------------------------------
        LDA     CMDPRV
        CMP     #CMD_IDX.DEL
        BEQ     NPWD_DONE
        CMP     #CMD_IDX.RENAME
        BEQ     NPWD_DONE
        CMP     #CMD_IDX.NCOPY
        BEQ     NPWD_DONE

NPWD_LOOP:
        LDA     #<RBUF
        LDY     #>RBUF
        JSR     PRINT_STRING
```

```
        LDA     #<STADCB
        LDY     #>STADCB
        JSR     DOSIOV
        JSR     PRINT_ERROR

    ;--------------------------------------
    ; Loop if more data to read
    ;--------------------------------------
        LDA     DVSTAT+1    ; Was there more to read (that is, was hi>0)?
        BNE     NPWD_LOOP   ; If yes, then do it again

NPWD_DONE:
        RTS

PWDDCB:
        .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   $30         ; DCOMND
        .BYTE   $40         ; DSTATS
        .BYTE   <RBUF       ; DBUFL
        .BYTE   >RBUF       ; DBUFH
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $00         ; DBYTL
        .BYTE   $01         ; DBYTH
        .BYTE   $00         ; DAUX1
        .BYTE   $00         ; DAUX2

; End of DO_NPWD
;--------------------------------------


AUTORUN_APPKEY:
        .WORD   $DB79               ; creator ID
        .BYTE   $00                 ; app ID
        .BYTE   $00                 ; key ID
        .BYTE   $00                 ; read or write mode
        .BYTE   $00                 ; unused

APPKEYCLOSEDCB:
        .BYTE   $70                 ; DDEVIC
        .BYTE   $01                 ; DUNIT
        .BYTE   $DB                 ; DCOMND
        .BYTE   $00                 ; DSTATS
        .BYTE   $00                 ; DBUFL
        .BYTE   $00                 ; DBUFH
        .BYTE   $0F                 ; DTIMLO
        .BYTE   $00                 ; DRESVD
        .BYTE   $00                 ; DBYTL
        .BYTE   $00                 ; DBYTH
        .BYTE   $00                 ; DAUX1
        .BYTE   $00                 ; DAUX2

APPKEYOPENDCB:
        .BYTE   $70                 ; DDEVIC
        .BYTE   $01                 ; DUNIT
        .BYTE   $DC                 ; DCOMND
        .BYTE   $80                 ; DSTATS
        .BYTE   <AUTORUN_APPKEY     ; DBUFL
        .BYTE   >AUTORUN_APPKEY     ; DBUFH
        .BYTE   $0F                 ; DTIMLO
        .BYTE   $00                 ; DRESVD
        .BYTE   $06                 ; DBYTL
        .BYTE   $00                 ; DBYTH
        .BYTE   $00                 ; DAUX1
        .BYTE   $00                 ; DAUX2

APPKEYREADDCB:
        .BYTE   $70                 ; DDEVIC
        .BYTE   $01                 ; DUNIT
        .BYTE   $DD                 ; DCOMND
        .BYTE   $40                 ; DSTATS
        .BYTE   <LNBUF              ; DBUFL
        .BYTE   >LNBUF              ; DBUFH
        .BYTE   $01                 ; DTIMLO - minimize timeout
        .BYTE   $00                 ; DRESVD
        .BYTE   MAX_APPKEY_LEN+2    ; DBYTL (+2 for # bytes)
        .BYTE   $00                 ; DBYTH
        .BYTE   $00                 ; DAUX1
        .BYTE   $00                 ; DAUX2

APPKEYWRITEDCB:
        .BYTE   $70                 ; DDEVIC
        .BYTE   $01                 ; DUNIT
        .BYTE   $DE                 ; DCOMND
        .BYTE   $80                 ; DSTATS
        .BYTE   $FF                 ; DBUFL
        .BYTE   $05                 ; DBUFH (expect page 5)
        .BYTE   $0F                 ; DTIMLO
        .BYTE   $00                 ; DRESVD
        .BYTE   MAX_APPKEY_LEN      ; DBYTL
        .BYTE   $00                 ; DBYTH
        .BYTE   $FF                 ; DAUX1 (# actual bytes)
        .BYTE   $00                 ; DAUX2

;--------------------------------------
SUBMIT_AUTORUN:
;--------------------------------------
    ; At initial DOS boot, read URL for
    ; app key file from SD card's
    ; FujiNet folder.
    ;
    ; filename: db790000.key
    ; contents: url to a batch file
    ;--------------------------------------
        JSR     LDBUFA

    ; Open app key
        LDA     #$00            ; Open for read
        STA     AUTORUN_APPKEY+4
        LDA     #<APPKEYOPENDCB
        LDY     #>APPKEYOPENDCB
        JSR     DOSIOV

        CPY     #$01            ; Was open successful?
        BEQ     AUTOSUB_NEXT    ; Yes. Continue.
        RTS                     ; No. Exit

AUTOSUB_NEXT:
    ; Read app key
        LDA     #<APPKEYREADDCB
        LDY     #>APPKEYREADDCB
        JSR     DOSIOV

    ; Close app key
        LDA     #<APPKEYCLOSEDCB
        LDY     #>APPKEYCLOSEDCB
        JSR     DOSIOV

    ; Does the returned URL contain something?
        LDX     LNBUF           ; X contains strlen of AUTORUN path
        BNE     AUTORUN_CALL_SUBMIT


AUTOSUB_DONE:
        RTS

AUTORUN_CALL_SUBMIT:
    ; Replace end-of-line in buffer with null terminator
        DEX                     ; Move index back 1 position
        LDA     #$00            ;
        STA     LNBUF+2,X       ; Write null-terminator
        LDA     #$02            ; Change arg1 location...
        STA     CMDSEP          ;  to point to filename

    ;--------------------------------------
    ; If here because of "AUTORUN ?", then
    ; print contents of appkey file. But first
    ; we have to terminate appkey string with EOL
    ;--------------------------------------
        LDA     AUTORUN_QUERY_FLG
        CMP     #'?'
        BNE     SUBMIT_NEXT1

        LDA     #EOL            ; Inject EOL to terminate string
        STA     LNBUF+2,X
        LDA     #<(LNBUF+2)
        LDY     #>(LNBUF+2)
        JMP     PRINT_STRING    ; Print AUTORUN path and sneak out

;--------------------------------------
DO_SUBMIT:
;--------------------------------------
        LDA     CMDSEP
        BNE     SUBMIT_NEXT1

    ; Filename required
        LDA     #<MISSING_FILE_STR
        LDY     #>MISSING_FILE_STR
        JMP     PRINT_STRING

SUBMIT_NEXT1:
    ; Default to NOSCREEN
        LDA     #$00
        STA     ECHO_FLG

    ; Prep file path
        JSR     GET_DOSDR       ; Get DUNIT
        JSR     PREPEND_DRIVE

    ; OPEN #1, 4, 0, file path
        JSR     NEXT_IOCB       ; X will be like $10
        BCC     @+              ; Skip ahead if success
        RTS                     ; Quit if no available channels

@:      STX     SUBMIT_IOCB
        LDY     #OINPUT         ; Open for input
        JSR     CIOOPEN         ; Open filename @ (INBUFF)
        BPL     SUBMIT_NEXT2
        JMP     PRINT_ERROR

    ; Read batch file character by character
    ; This allows it be end-of-line agnostic
    ; Branch forward when an end-of-line is interpretted.

SUBMIT_NEXT2:
        JSR     LDBUFA      ; Reset INBUFF to $0582
        DEC     INBUFF      ; Init 1 byte before buffer
        LDA     #$FF        ; Clear command
        STA     CMD

SUBMIT_GETCH:
        INC     INBUFF          ; Advance pointer
        BNE     SUBMIT_NEXT3
        INC     INBUFF+1

SUBMIT_NEXT3:
        LDX     SUBMIT_IOCB     ; X will be like $10
        LDA     #$01            ; Get 1 byte
        LDY     #$00            ; ditto

        JSR     CIOGET          ; Get byte from file
        LDY     #$00            ;
        LDA     (INBUFF),Y      ; byte will be here

        CMP     #CR             ; Just skip if Windows CR
        BEQ     SUBMIT_GETCH

        CMP     #LF             ; Convert LF to EOL
        BNE     SUBMIT_EOL
        LDA     #EOL
        STA     (INBUFF),Y

SUBMIT_EOL:
        CMP     #EOL            ; At end of command line?
        BNE     SUBMIT_GETCH    ; No. Get next byte.

    ; Here if we've reached the end of a command line.
    ; At end of file?
        LDX     SUBMIT_IOCB     ; X will be like $10
        LDA     ICSTA,X         ; Inspect return code
        CMP     #EOF
        BEQ     SUBMIT_DONE     ; No error, try parsing cmd

        LDA     ECHO_FLG        ; Skip echo if SCREEN is disabled
        BEQ     SUBMIT_NEXT4
        LDA     LNBUF
        CMP     #'@'            ; Skip lines beginning with @
        BEQ     SUBMIT_NEXT4

    ; Echo commands
        JSR     LDBUFA
        LDA     INBUFF
        LDY     INBUFF+1
        JSR     PRINT_STRING

SUBMIT_NEXT4:
        JSR     GETCMDTEST
        JSR     PARSECMD
        JSR     DOCMD
        SEC
        BCS     SUBMIT_NEXT2

SUBMIT_DONE:
        LDX     #$10
        JMP     CIOCLOSE

SUBMIT_IOCB:
        .BYTE   $00             ; Place to store channel

; End of DO_SUBMIT
```

```
;---------------------------------------

;---------------------------------------
DO_TYPE:
;---------------------------------------
        LDA     CMDSEP
        BNE     TYPE_SKIP

TYPE_USAGE:
        LDA     #<MISSING_FILE_STR
        LDY     #>MISSING_FILE_STR
        JMP     PRINT_STRING

TYPE_SKIP:
        JSR     GET_DOSDR       ; Get DUNIT
        JSR     PREPEND_DRIVE

    ; Open input file
        JSR     NEXT_IOCB       ; Get next available IOCB channel
        BCC     @+              ; Carry is set on failure
        RTS                     ; Unable to find available channel

@:      STX     TYPE_IOCB       ; X will be like $10
        LDY     #$04            ; Open for input
        JSR     CIOOPEN         ; Open filename @ (INBUFF)
        BPL     TYPE_NEXT

    ; If open failed, Print error
        LDX     #$10            ; File #1
        LDY     ICSTA,X
        JMP     PRINT_ERROR

TYPE_NEXT:
    ; Initialize pagination
        JSR     DO_CLS
        LDA     #21
        STA     SCRFLG

TYPE_LOOP:
    ; Bail if ESC key is pressed
        LDA     CH
        CMP     #ESC_KEY
        BEQ     TYPE_DONE

    ; Check if page is full
        LDA     SCRFLG
        CMP     #22             ; if SCRFLG < 21
        BCC     TYPE_READ       ; then skip to read

    ; Here if page is full
    ; Wait for keypress
        LDA     #$FF            ; Clear keypress
        STA     CH

TYPE_WAIT:
        LDX     CH              ; Will be $FF if no keypress
        INX                     ; $FF --> $00
        BEQ     TYPE_WAIT       ; Keep waiting if $00

        CPX     #ESC_KEY        ; Leave if ESC key pressed
        BEQ     TYPE_DONE

    ; Reset pagination
        LDA     #$00
        STA     SCRFLG

TYPE_READ:
    ; Read from file
;        LDX     #$10            ; IOCB offset (channel)
        LDX     TYPE_IOCB
        LDA     #$01            ; ICBLL (buffer length lo) Request 1 byte
        LDY     #$00            ; ICBLH (buffer length hi)
        JSR     CIOGET

    ; Quit if EOF
;        LDX     #$10
        LDX     TYPE_IOCB
        LDA     ICSTA,X
        CMP     #EOF
        BEQ     TYPE_DONE

    ; Convert CRLF or LF --> EOL
        LDY     #$00
        LDA     (INBUFF),Y
        CMP     #CR     ; Skip CR
        BEQ     TYPE_NEXT3
        CMP     #LF     ; Convert LF --> EOL
        BNE     TYPE_NEXT2
        LDA     #EOL
        STA     (INBUFF),Y

TYPE_NEXT2:
    ; Write to screen
        LDX     #$00
        LDA     #$01
        LDY     #$00
        JSR     CIOPUT

TYPE_NEXT3:
    ; Do next
        JMP     TYPE_LOOP

TYPE_DONE:
        LDA     #$FF
        STA     CH
;        LDX     #$10            ; Close File #1
        LDX     TYPE_IOCB
        JMP     CIOCLOSE        ;

TYPE_OPEN_ERR_STR:
        .BYTE   'UNABLE TO OPEN FILE',EOL

TYPE_IOCB:
        .BYTE   $00             ; IOCB Channel for open file

;---------------------------------------
DO_CAR:
;---------------------------------------

    ;-----------------------------------
    ; Is cart address space RAM or ROM?
    ;-----------------------------------
        JSR     CHECK_IF_ROM    ; returns Y=1 -> ROM. Y=0 -> RAM.
        TYA                     ; Xfer affects Z flag
        BNE     DO_CAR_NEXT     ; if Y=1 (ROM) skip ahead

    ;-----------------------------------
    ; RAM found
    ;-----------------------------------
        LDA     #<DO_CAR_ERR
        LDY     #>DO_CAR_ERR
        JMP     PRINT_STRING    ; Print error and bail

DO_CAR_NEXT:
    ;-----------------------------------
    ; Border used to indicate program in ROM
    ; Revert border before returning to ROM
    ;-----------------------------------
        LDA     COLOR4_ORIG
        STA     COLOR4          ; Reset border to orig color

    ;-----------------------------------
    ; Warmstart
    ;-----------------------------------
        LDA     #$FF
        STA     $08
        JMP     ($BFFA)         ; Bye

DO_CAR_ERR:
        .BYTE   'NO CARTRIDGE',EOL

;---------------------------------------
DO_CLS:
;---------------------------------------
        LDA     #<CLS_STR
        LDY     #>CLS_STR
        JMP     PRINT_STRING

CLS_STR:
        .BYTE   125,EOL

;---------------------------------------
DO_MENU:
;---------------------------------------
    ; The MENU command (from the CLI): (re)load and run the menu
    ; module.  MENU_LAUNCH resets the stack, so we can be several
    ; frames deep inside CP's command loop and still return cleanly.
        JMP     MENU_LAUNCH

;---------------------------------------
; MENU_LAUNCH: (re)load the menu module and draw the full menu.
; Used at boot, for the MENU command, and after RETURN.  Resets the
; stack (top-level command-processor entry), reloads the module, then
; runs it from the top (MENU_CP clears the screen and redraws).
; The menu is NOT resident, which is what keeps MEMLO low.
;---------------------------------------
MENU_LAUNCH:
        LDX     #$FF            ; top-level (re)entry: reset the stack
        TXS
        JSR     MENU_LOAD
        JMP     MENU_RUN        ; MENU_CP: clear screen + draw full menu

;---------------------------------------
; Resident trampolines for menu-item dispatch.  The menu module
; assembles the command line into LNBUF, then JMPs here.  Running
; the pipeline from RESIDENT code means a command that overwrites
; the transient menu module can't corrupt our return path.  After the
; command we reload the module and jump to MENU_SELECT (NOT MENU_CP),
; so the command's output stays on screen under a fresh prompt.
;---------------------------------------
MENU_DISP:                      ; keyword items: parse the assembled LNBUF
        LDA     #$FF
        STA     CMD
        JSR     GETCMDTEST
        JSR     PARSECMD
MENU_DISP2:                     ; drive item enters here (CMD preset)
        JSR     DOCMD
        LDX     #$FF            ; reset stack after the command
        TXS
        JSR     MENU_LOAD       ; restore the (possibly clobbered) module
        JMP     MENU_SELECT     ; re-prompt, keeping command output visible

;---------------------------------------
; MENU_LOAD: read MENU_CNT sectors from the ATR (sector MENU_SECT)
; into MENU_RUN, then RTS.  Modeled on DO_OVERLAY, but the module is
; loaded to its own run address (assembled there -> no relocation).
;---------------------------------------
MENU_LOAD:
        LDA     #MENU_SECT      ; first ATR sector of the module
        STA     GET_SECTOR_DCB+DCB_IDX.DAUX1
        LDA     #$00
        STA     GET_SECTOR_DCB+DCB_IDX.DAUX2
        LDA     #MENU_CNT       ; number of sectors
        STA     SECT_CNT
        LDA     #<MENU_RUN      ; destination = module run address
        STA     GET_SECTOR_DCB+DCB_IDX.DBUFL
        LDA     #>MENU_RUN
        STA     GET_SECTOR_DCB+DCB_IDX.DBUFH
MENU_LOAD_LOOP:
        LDA     #<GET_SECTOR_DCB
        LDY     #>GET_SECTOR_DCB
        JSR     DOSIOV
        DEC     SECT_CNT
        BEQ     MENU_LOAD_DONE
        INC     GET_SECTOR_DCB+DCB_IDX.DAUX1     ; next sector (<256)
        CLC
        LDA     GET_SECTOR_DCB+DCB_IDX.DBUFL     ; dest += one sector
        ADC     #SECTOR_SIZE
        STA     GET_SECTOR_DCB+DCB_IDX.DBUFL
        BCC     MENU_LOAD_LOOP
        INC     GET_SECTOR_DCB+DCB_IDX.DBUFH
        JMP     MENU_LOAD_LOOP
MENU_LOAD_DONE:
        RTS

;---------------------------------------
; "P" item: a persistent classic CLI.  Resident, so a command that
; clobbers the menu module can't break this loop.  Typing MENU
; (DO_MENU) returns to the menu.
;---------------------------------------
MENU_CLI:
        JSR     CP              ; Nn: prompt, read + dispatch
        JMP     MENU_CLI

;---------------------------------------
DO_COLD:
;---------------------------------------
        JMP     COLDSV

;---------------------------------------
DO_NOSCREEN:
;---------------------------------------
        LDA     #$00
        STA     ECHO_FLG    ; Disable echo in batch processing
        RTS

;---------------------------------------
DO_SCREEN:
;---------------------------------------
        LDA     #$01
        STA     ECHO_FLG    ; Enable echo in batch processing
        RTS
```

```
;---------------------------------------
DO_PRINT:
;---------------------------------------
        LDA     CMDSEP
        BEQ     PRINT_DONE

        CLC
        ADC     INBUFF
        LDY     INBUFF+1
        JMP     PRINT_STRING

PRINT_DONE:
        RTS

;---------------------------------------
DO_REM:
;---------------------------------------
        RTS

;---------------------------------------
DO_RUN:
;---------------------------------------
        LDA     CMDSEP      ; Get position for address arg
        TAY                 ; Offset to arg used later
        CLC
        ADC     #$04
        STA     RBUF

        JSR     ASCII2ADDR  ; Convert text to an addr
        BCS     DO_REM      ; Re-use nearby RTS

        JMP     (INBUFF)    ;

;---------------------------------------
DO_WARM:
;---------------------------------------
        JMP     WARMSV

;---------------------------------------
REMOUNT_DRIVE:
;---------------------------------------

    ;---------------------------------------
    ; Workaround for timeout issue regarding idempotent commands that
    ; unmount the server.  So far, these are DEL and RENAME. This
    ; routine, remounts the TNFS URL by calling NPWD and attempts
    ; a MKDIR on the returned mount point. Hopefully this is an
    ; non-consequential operation since the directory already exists.
    ;---------------------------------------

        JSR     DO_NPWD         ; Curr dir for drive now in RBUF

        LDA     RBUF            ; Quit if not TNFS. Only TNFS is affected.
        CMP     #'T'            ; TODO More letters needed if...
        BNE     REMOUNT_DONE    ; ...another Txxx protocol exists

        LDA     #'N'
        STA     RBUF+0
        LDA     DOSDR           ; Get drive number
        ORA     #'0'            ; Convert, say, 1 to '1'
        STA     RBUF+1
        LDA     #':'
        STA     RBUF+2

        LDA     #CMD_MKDIR
        STA     GENDCB+DCB_IDX.DCOMND
        LDA     #<RBUF                  ; TODO Is this needed
        STA     GENDCB+DCB_IDX.DBUFL    ; TODO or is it hardcoded in DO_GENERIC?
        LDA     #>RBUF
        STA     GENDCB+DCB_IDX.DBUFH

        LDA     #<GENDCB
        LDY     #>GENDCB
        JMP     DOSIOV

REMOUNT_DONE:
        RTS

;---------------------------------------
PREPEND_DRIVE:
;---------------------------------------
        ; Inject "Nn:" in front of a plain filename
        ; before passing it to the FujiNet
        LDY     #$00
        LDA     #'N'
        CMP     (INBUFF),Y  ; Does arg1 already begin with N?
        BNE     PREPEND_DRIVE_DONE_NEXT_2

    ; Check if 2nd char is : as in N:
        INY
        LDA     #':'
        CMP     (INBUFF),Y
        BNE     PREPEND_DRIVE_DONE_NEXT_1

    ; Here if N:
    ; Change to Nn: where n is prompt digit char such as 2 in N2:
        DEC     INBUFF
        LDY     #$00
        LDA     #'N'
        STA     (INBUFF),Y

        INY
        LDA     PRMPT+2
        STA     (INBUFF),Y
        JMP     PREPEND_DRIVE_DONE

    ; Here if 2nd char is not :
    ; Check if 3rd char is :
PREPEND_DRIVE_DONE_NEXT_1:
        INY
        CMP     (INBUFF),Y
        BEQ     PREPEND_DRIVE_DONE  ; Done if 3rd char is :

    ; Otherwise inject Nn: into filename
    ; Move input buffer pointer back 3 bytes
PREPEND_DRIVE_DONE_NEXT_2:
        SEC
        LDA     INBUFF
        SBC     #$03
        STA     INBUFF
        LDA     INBUFF+1
        SBC     #$00
        STA     INBUFF+1

        ; Inject PRMPT to front of arg1
        LDY     #$03
PREPEND_DRIVE_LOOP:
        LDA     PRMPT,Y
        DEY
        STA     (INBUFF),Y
        BNE     PREPEND_DRIVE_LOOP

PREPEND_DRIVE_DONE:
        LDY     #$01
        RTS             ; Y = $00 here

;;---------------------------------------
;PREPEND_DRIVE:
;;---------------------------------------
;        ; Inject "Nn:" in front of a plain filename
;        ; before passing it to the FujiNet
;        LDY     #$00
;        LDA     #'N'
;        CMP     (INBUFF),Y  ; Does arg1 already begin with N?
;
;        LDY     #$02
;        LDA     #':'
;        CMP     (INBUFF),Y
;        BEQ     PREPEND_DRIVE_DONE
;        DEY
;        CMP     (INBUFF),Y
;        BEQ     PREPEND_DRIVE_DONE
;
;        ; Move input buffer pointer back 3 bytes
;        SEC
;        LDA     INBUFF
;        SBC     #$03
;        STA     INBUFF
;        LDA     INBUFF+1
;        SBC     #$00
;        STA     INBUFF+1
;
;        ; Inject PRMPT to front of arg1
;        LDY     #$03
;PREPEND_DRIVE_LOOP:
;        LDA     PRMPT,Y
;        DEY
;        STA     (INBUFF),Y
;        BNE     PREPEND_DRIVE_LOOP
;
;PREPEND_DRIVE_DONE:
;        LDY     #$01
;        RTS             ; Y = $00 here

;;---------------------------------------
;APPEND_SLASH:
;;---------------------------------------
;    ;---------------------------------------
;    ; Skip if relative path (..)
;    ;---------------------------------------
;        LDY     #$00
;        LDA     #'.'
;        CMP     (INBUFF),Y
;        BEQ     APPEND_SLASH_DONE
;
;        LDY     #$FF        ; Iterate thru arg2 until EOF
;APPEND_SLASH_LOOP:
;        INY                 ; Zero on 1st pass
;        LDA     (INBUFF),Y
;        CMP     #EOL
;        BNE     APPEND_SLASH_LOOP
;
;        DEY                 ; Move pointer back one character
;        LDA     (INBUFF),Y
;        CMP     #'/'        ; If already slash then skip rest
;        BEQ     APPEND_SLASH_DONE
;        CMP     #':'        ; If a drive, skip
;        BEQ     APPEND_SLASH_DONE
;
;        INY                 ; Else inject '/' + EOL
;        LDA     #'/'
;        STA     (INBUFF),Y
;        INY
;        LDA     #EOL
;        STA     (INBUFF),Y
;
;APPEND_SLASH_DONE:
;        RTS


PRMPT:
        .BYTE   EOL,'N :'

;;; End CP ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Variables

NTRDCB:
        .BYTE   DEVIDN  ; DDEVIC
        .BYTE   $FF     ; DUNIT
        .BYTE   'T'     ; DCOMND
        .BYTE   $00     ; DSTATS
        .BYTE   $00     ; DBUFL
        .BYTE   $00     ; DBUFH
        .BYTE   $0F     ; DTIMLO
        .BYTE   $00     ; DRESVD
        .BYTE   $00     ; DBYTL
        .BYTE   $00     ; DBYTH
        .BYTE   $00     ; DAUX1
        .BYTE   $00     ; DAUX2

        .ENUM   DCB_IDX
        ;---------------
                DDEVIC              ; 0
                DUNIT               ; 1
                DCOMND              ; 2
                DSTATS              ; 3
                DBUFL               ; 4
                DBUFH               ; 5
                DTIMLO              ; 6
                DRESVD              ; 7
                DBYTL               ; 8
                DBYTH               ; 9
                DAUX1               ; 10
                DAUX2               ; 11
        .ENDE

        .ENUM   CMD_IDX
        ;---------------
                NCD                 ;  0
                DIR                 ;  1
                DEL                 ;  2
                LOAD                ;  3
                MKDIR               ;  4
                NCOPY               ;  5
                NPWD                ;  6
                NTRANS              ;  7
                PASS                ;  8
                RENAME              ;  9
                RMDIR               ; 10
                SAVE                ; 11
                SUBMIT              ; 12
```

```
                TYPE            ; 13
                USER            ; 14
                AUTORUN         ; 15
                BASIC           ; 16
                CAR             ; 17
                CLS             ; 18
                COLD            ; 19
                DUMP            ; 20
                FILL            ; 21
                HELP            ; 22
                NOSCREEN        ; 23
                PRINT           ; 24
                REENTER         ; 25
                REM             ; 26
                RUN             ; 27
                SCREEN          ; 28
                WARM            ; 29
                XEP             ; 30
                DRIVE_CHG       ; 31
                MENU            ; 32
        .ENDE

CMD_DCOMND:
        .BYTE   CMD_CD          ;  0 NCD
        .BYTE   CMD_DIR         ;  1 DIR
        .BYTE   CMD_DEL         ;  2 DEL
        .BYTE   CMD_LOAD        ;  3 LOAD
        .BYTE   CMD_MKDIR       ;  4 MKDIR
        .BYTE   CMD_NCOPY       ;  5 NCOPY
        .BYTE   CMD_NPWD        ;  6 NPWD
        .BYTE   CMD_NTRANS      ;  7 NTRANS
        .BYTE   CMD_PASS        ;  8 PASS
        .BYTE   CMD_RENAME      ;  9 RENAME
        .BYTE   CMD_RMDIR       ; 10 RMDIR
        .BYTE   CMD_SAVE        ; 11 SAVE
        .BYTE   CMD_SUBMIT      ; 12 SUBMIT
        .BYTE   CMD_TYPE        ; 13 TYPE
        .BYTE   CMD_USER        ; 14 USER
        .BYTE   CMD_AUTORUN     ; 15 AUTORUN
        .BYTE   CMD_BASIC       ; 16 BASIC
        .BYTE   CMD_CAR         ; 17 CAR
        .BYTE   CMD_CLS         ; 18 CLS
        .BYTE   CMD_COLD        ; 19 COLD
        .BYTE   CMD_DUMP        ; 20 DUMP
        .BYTE   CMD_FILL        ; 21 FILL
        .BYTE   CMD_HELP        ; 22 HELP
        .BYTE   CMD_NOSCREEN    ; 23 NOSCREEN
        .BYTE   CMD_PRINT       ; 24 PRINT
        .BYTE   CMD_REENTER     ; 25 REENTER
        .BYTE   CMD_REM         ; 26 REM
        .BYTE   CMD_RUN         ; 27 RUN
        .BYTE   CMD_SCREEN      ; 28 SCREEN
        .BYTE   CMD_WARM        ; 29 WARM
        .BYTE   CMD_XEP         ; 30 XEP
        .BYTE   CMD_DRIVE_CHG   ; 31
        .BYTE   CMD_MENU        ; 32 MENU

COMMAND:
        .CB     "NCD"           ;  0 NCD
        .BYTE   CMD_IDX.NCD

        .CB     "DIR"           ;  1 DIR
        .BYTE   CMD_IDX.DIR

        .CB     "DEL"           ;  2 DEL
        .BYTE   CMD_IDX.DEL

        .CB     "LOAD"          ;  3 LOAD
        .BYTE   CMD_IDX.LOAD

        .CB     "MKDIR"         ;  4 MKDIR
        .BYTE   CMD_IDX.MKDIR

        .CB     "NCOPY"         ;  5 NCOPY
        .BYTE   CMD_IDX.NCOPY

        .CB     "NPWD"          ;  6 NPWD
        .BYTE   CMD_IDX.NPWD

        .CB     "NTRANS"        ;  7 NTRANS
        .BYTE   CMD_IDX.NTRANS

        .CB     "PASS"          ;  8 PASS
        .BYTE   CMD_IDX.PASS

        .CB     "RENAME"        ;  9 RENAME
        .BYTE   CMD_IDX.RENAME

        .CB     "RMDIR"         ; 10 RMDIR
        .BYTE   CMD_IDX.RMDIR

        .CB     "SAVE"          ; 11 SAVE
        .BYTE   CMD_IDX.SAVE

        .CB     "SUBMIT"        ; 12 SUBMIT
        .BYTE   CMD_IDX.SUBMIT

        .CB     "TYPE"          ; 13 TYPE
        .BYTE   CMD_IDX.TYPE

        .CB     "USER"          ; 14 USER
        .BYTE   CMD_IDX.USER

        .CB     "AUTORUN"       ; 15 AUTORUN
        .BYTE   CMD_IDX.AUTORUN

        .CB     "BASIC"         ; 16 NOBASIC
        .BYTE   CMD_IDX.BASIC

        .CB     "CAR"           ; 17 CAR
        .BYTE   CMD_IDX.CAR

        .CB     "CLS"           ; 18 CLS
        .BYTE   CMD_IDX.CLS

        .CB     "COLD"          ; 19 COLD
        .BYTE   CMD_IDX.COLD

        .CB     "DUMP"          ; 20 DUMP
        .BYTE   CMD_IDX.DUMP

        .CB     "FILL"          ; 21 FILL
        .BYTE   CMD_IDX.FILL

        .CB     "HELP"          ; 23 HELP
        .BYTE   CMD_IDX.HELP

        .CB     "@NOSCREEN"     ; 24 @NOSCREEN
        .BYTE   CMD_IDX.NOSCREEN

        .CB     "PRINT"         ; 25 PRINT
        .BYTE   CMD_IDX.PRINT

        .CB     "REENTER"       ; 26 REENTER
        .BYTE   CMD_IDX.REENTER

        .CB     "REM"           ; 27 REM
        .BYTE   CMD_IDX.REM

        .CB     "RUN"           ; 28 RUN
        .BYTE   CMD_IDX.RUN

        .CB     "@SCREEN"       ; 29 @SCREEN
        .BYTE   CMD_IDX.SCREEN

        .CB     "WARM"          ; 30 WARM
        .BYTE   CMD_IDX.WARM

        .CB     "XEP"           ; 31 XEP
        .BYTE   CMD_IDX.XEP

        .CB     "MENU"          ; 32 MENU (leave CLI, return to menu)
        .BYTE   CMD_IDX.MENU

        ; Drive Change intentionally omitted

; Aliases
        .CB     "CD"            ; CD = NCD
        .BYTE   CMD_IDX.NCD

        .CB     "CWD"           ; CWD = NCD
        .BYTE   CMD_IDX.NCD

        .CB     "COPY"          ; COPY = NCOPY
        .BYTE   CMD_IDX.NCOPY

        .CB     "ERASE"         ; ERASE = DEL
        .BYTE   CMD_IDX.DEL

        .CB     "ERA"           ; ERA = DEL
        .BYTE   CMD_IDX.DEL

        .CB     "X"             ; X = LOAD
        .BYTE   CMD_IDX.LOAD

        .CB     "PWD"           ; PWD = NPWD
        .BYTE   CMD_IDX.NPWD

        .CB     "REE"           ; R = REENTER
        .BYTE   CMD_IDX.REENTER

        .CB     "REN"           ; REN = RENAME
        .BYTE   CMD_IDX.RENAME

        .CB     "@"             ; @ = SUBMIT
        .BYTE   CMD_IDX.SUBMIT

        .CB     "#"             ; # = REM
        .BYTE   CMD_IDX.REM

        .CB     "'"             ; ' = REM
        .BYTE   CMD_IDX.REM

        ; With U1MB, a non-BASIC program might reside
        ; in ROM, then BASIC and NOBASIC feel awkward.
        ; So, ROMON and ROMOFF. (I know. Inconsistent.)

        .CB     "ROM"           ; ROMON = BASIC
        .BYTE   CMD_IDX.BASIC

COMMAND_SIZE = * - COMMAND - 1
        .BYTE   $FF

CMD_TAB_L:
        .BYTE   <(DO_GENERIC-1)     ;  0 NCD
        .BYTE   <(DO_DIR-1)         ;  1 DIR
        .BYTE   <(DO_NDEL-1)        ;  2 DEL (scan overlay -> single or wildcard)
        .BYTE   <(DO_LOAD-1)        ;  3 LOAD
        .BYTE   <(DO_GENERIC-1)     ;  4 MKDIR
        .BYTE   <(DO_NCOPY-1)       ;  5 NCOPY
        .BYTE   <(DO_NPWD-1)        ;  6 NPWD
        .BYTE   <(DO_NTRANS-1)      ;  7 NTRANS
        .BYTE   <(DO_GENERIC-1)     ;  8 PASS
        .BYTE   <(DO_GENERIC-1)     ;  9 RENAME
        .BYTE   <(DO_GENERIC-1)     ; 10 RMDIR
        .BYTE   <(DO_SAVE-1)        ; 11 SAVE
        .BYTE   <(DO_SUBMIT-1)      ; 12 SUBMIT
        .BYTE   <(DO_TYPE-1)        ; 13 TYPE
        .BYTE   <(DO_GENERIC-1)     ; 14 USER
        .BYTE   <(DO_AUTORUN-1)     ; 15 AUTORUN
        .BYTE   <(DO_BASIC-1)       ; 16 BASIC
        .BYTE   <(DO_CAR-1)         ; 17 CAR
        .BYTE   <(DO_CLS-1)         ; 18 CLS
        .BYTE   <(DO_COLD-1)        ; 19 COLD
        .BYTE   <(DO_DUMP-1)        ; 20 DUMP
        .BYTE   <(DO_FILL-1)        ; 21 FILL
        .BYTE   <(DO_HELP-1)        ; 22 HELP
        .BYTE   <(DO_NOSCREEN-1)    ; 23 NOSCREEN
        .BYTE   <(DO_PRINT-1)       ; 24 PRINT
        .BYTE   <(DO_REENTER-1)     ; 25 REENTER
        .BYTE   <(DO_REM-1)         ; 26 REM
        .BYTE   <(DO_RUN-1)         ; 27 RUN
        .BYTE   <(DO_SCREEN-1)      ; 28 SCREEN
        .BYTE   <(DO_WARM-1)        ; 29 WARM
        .BYTE   <(DO_XEP-1)         ; 30 XEP
        .BYTE   <(DO_DRIVE_CHG-1)   ; 31
        .BYTE   <(DO_MENU-1)        ; 32 MENU

CMD_TAB_H:
        .BYTE   >(DO_GENERIC-1)     ;  0 NCD
        .BYTE   >(DO_DIR-1)         ;  1 DIR
        .BYTE   >(DO_NDEL-1)        ;  2 DEL (scan overlay -> single or wildcard)
        .BYTE   >(DO_LOAD-1)        ;  3 LOAD
        .BYTE   >(DO_GENERIC-1)     ;  4 MKDIR
        .BYTE   >(DO_NCOPY-1)       ;  5 NCOPY
        .BYTE   >(DO_NPWD-1)        ;  6 NPWD
        .BYTE   >(DO_NTRANS-1)      ;  7 NTRANS
        .BYTE   >(DO_GENERIC-1)     ;  8 PASS
        .BYTE   >(DO_GENERIC-1)     ;  9 RENAME
        .BYTE   >(DO_GENERIC-1)     ; 10 RMDIR
        .BYTE   >(DO_SAVE-1)        ; 11 SAVE
        .BYTE   >(DO_SUBMIT-1)      ; 12 SUBMIT
        .BYTE   >(DO_TYPE-1)        ; 13 TYPE
        .BYTE   >(DO_GENERIC-1)     ; 14 USER
        .BYTE   >(DO_AUTORUN-1)     ; 15 AUTORUN
        .BYTE   >(DO_BASIC-1)       ; 16 BASIC
        .BYTE   >(DO_CAR-1)         ; 17 CAR
        .BYTE   >(DO_CLS-1)         ; 18 CLS
        .BYTE   >(DO_COLD-1)        ; 19 COLD
        .BYTE   >(DO_DUMP-1)        ; 20 DUMP
        .BYTE   >(DO_FILL-1)        ; 21 FILL
        .BYTE   >(DO_HELP-1)        ; 22 HELP
```

```
        .BYTE   >(DO_NOSCREEN-1)    ; 23 NOSCREEN
        .BYTE   >(DO_PRINT-1)       ; 24 PRINT
        .BYTE   >(DO_REENTER-1)     ; 25 REENTER
        .BYTE   >(DO_REM-1)         ; 26 REM
        .BYTE   >(DO_RUN-1)         ; 27 RUN
        .BYTE   >(DO_SCREEN-1)      ; 28 SCREEN
        .BYTE   >(DO_WARM-1)        ; 29 WARM
        .BYTE   >(DO_XEP-1)         ; 30 XEP
        .BYTE   >(DO_DRIVE_CHG-1)   ; 31
        .BYTE   >(DO_MENU-1)        ; 32 MENU

        ; Overlay tables

        .ENUM   OVL_IDX
                AUTORUN
                BASIC
                DIR
                DUMP
                FILL
                HELP
                NCOPY
                NCOPY1
                NCOPY2
                NTRANS
                REENTER
                SAVE
                XEP
                NDEL
        .ENDE

        ; Derive ATR Sector where code is stored
OVL_SECT_TAB_L:
        .BYTE   <(OVL_AUTORUN/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_BASIC/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_DIR/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_DUMP/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_FILL/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_HELP/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_NCOPY/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_NCOPY1/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_NCOPY2/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_NTRANS/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_REENTER/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_SAVE/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_XEP/SECTOR_SIZE-$0D)
        .BYTE   <(OVL_NDEL/SECTOR_SIZE-$0D)

;OVL_SECT_TAB_H:
;        .BYTE   >(OVL_AUTORUN/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_BASIC/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_DIR/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_DUMP/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_FILL/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_HELP/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_NCOPY/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_NCOPY1/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_NCOPY2/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_NTRANS/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_REENTER/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_SAVE/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_XEP/SECTOR_SIZE-$0D)
;        .BYTE   >(OVL_NDEL/SECTOR_SIZE-$0D)

        ; Derive number of ATR sectors used to store code
OVL_SECT_CNT_TAB:
        .BYTE   [END_OVL_AUTORUN-OVL_AUTORUN]/SECTOR_SIZE
        .BYTE   [END_OVL_BASIC-OVL_BASIC]/SECTOR_SIZE
        .BYTE   [END_OVL_DIR-OVL_DIR]/SECTOR_SIZE
        .BYTE   [END_OVL_DUMP-OVL_DUMP]/SECTOR_SIZE
        .BYTE   [END_OVL_FILL-OVL_FILL]/SECTOR_SIZE
        .BYTE   [END_OVL_HELP-OVL_HELP]/SECTOR_SIZE
        .BYTE   [END_OVL_NCOPY-OVL_NCOPY]/SECTOR_SIZE
        .BYTE   [END_OVL_NCOPY1-OVL_NCOPY1]/SECTOR_SIZE
        .BYTE   [END_OVL_NCOPY2-OVL_NCOPY2]/SECTOR_SIZE
        .BYTE   [END_OVL_NTRANS-OVL_NTRANS]/SECTOR_SIZE
        .BYTE   [END_OVL_REENTER-OVL_REENTER]/SECTOR_SIZE
        .BYTE   [END_OVL_SAVE-OVL_SAVE]/SECTOR_SIZE
        .BYTE   [END_OVL_XEP-OVL_XEP]/SECTOR_SIZE
        .BYTE   [END_OVL_NDEL-OVL_NDEL]/SECTOR_SIZE

        ; DEVHDL TABLE FOR N:
CIOHND  .WORD   OPEN-1
        .WORD   CLOSE-1
        .WORD   GET-1
        .WORD   PUT-1
        .WORD   STATUS-1
        .WORD   SPEC-1

      ; BANNERS

BREADY  .BYTE   '#FUJINET NOS v1.0.0',EOL
BERROR  .BYTE   '#FUJINET ERROR',EOL

        ; MESSAGES

CDERR   .BYTE   'Nn?',EOL

        ; STRING CONSTANTS

MISSING_FILE_STR:
        .BYTE   'FILE?',EOL

SAVE_ERROR_STR:
        .BYTE   'SAVE Nn:FILE,START,END[,INIT][,RUN]',EOL

SAVE_ADDR_FLG:
        .BYTE   %00000001
        .BYTE   %00000010
        .BYTE   %00000100
        .BYTE   %00001000
        .BYTE   %00010000

SAVE_HEADER:
        .BYTE   $FF,$FF,$00,$00,$00,$00,$00,$00
SAVE_INIT:
        .BYTE   $E2,$02,$E3,$02,$00,$00
SAVE_RUN:
        .BYTE   $E0,$02,$E1,$02,$00,$00


        ; VARIABLES

DOSDR           .BYTE   $01 ; DOS DRIVE
CMD             .BYTE   $01
CMDPRV          .BYTE   $01
OVLPRV          .BYTE   $FF ; Previous Overlay Command
ECHO_FLG        .BYTE   $01 ; Echo batch cmds (1=enabled,0=disabled)
AUTORUN_FLG     .BYTE   $00 ; Checked at DOS entry. Runs only on first pass
WRITEMODE       .BYTE   $00 ; Used for open, truncate or open, append
SOURCE_IOCB     .BYTE   $00 ; Used in NCOPY
TARGET_IOCB     .BYTE   $00 ; Used in NCOPY
MENU_KLEN       .BYTE   $00 ; Menu: assembled keyword length
QINQ            .BYTE   $00 ; QSTRIP: inside-double-quotes flag
WILD_CH         .BYTE   $00 ; Wildcard COPY: directory IOCB channel
WNPTR           .BYTE   $00,$00 ; Wildcard COPY: cursor into name list
WFROM           .BYTE   $00,$00 ; Wildcard COPY: prepended FROM pointer
WILD_MODE       .BYTE   $00 ; Wildcard driver select: 0=COPY, 1=DEL

TRIP            .BYTE   $01 ; INTR FLAG
RLEN    :MAXDEV .BYTE   $00 ; RCV LEN
ROFF    :MAXDEV .BYTE   $00 ; RCV OFFSET
TOFF    :MAXDEV .BYTE   $00 ; TRX OFFSET
INQDS           .BYTE   $01 ; DSTATS INQ

DVS2    :MAXDEV .BYTE   $00 ; DVSTAT+2 SAVE
DVS3    :MAXDEV .BYTE   $00 ; DVSTAT+3 SAVE

BWRAW           .BYTE   $00,$00 ; Uncapped bytes-waiting (burst)
CHUNK           .BYTE   $00,$00 ; Burst transfer size
DECR            .BYTE   $00,$00 ; Burst pointer/length decrement
NPBUF           .BYTE   $00,$00,$00 ; NOTE/POINT 24-bit position

COLOR4_ORIG     .BYTE   $00 ; Hold prev border color

       ; BUFFERS (PAGE ALIGNED)
        .ALIGN  $100, $00
BOOTEND:

; Overlay command RAM (2 sectors)
OVLBUF: :$100   .BYTE   $00

RBUF:   :$80    .BYTE   $00 ; 128 bytes
TBUF:   :$80    .BYTE   $00 ; 128 bytes

; Binary loader working variables
BAL     = RBUF
BAH     = RBUF+1    ;
STL     = TBUF      ; Payload Start address
STH     = TBUF+1
ENL     = TBUF+2    ; Payload End address
ENH     = TBUF+3
HEADL   = TBUF+4    ; Bytes read from existing cache
HEADH   = TBUF+5
BODYL   = TBUF+6    ; Total bytes read in contiguous 512-byte blocks
BODYH   = TBUF+7
BLL     = TBUF+8    ; Payload Buffer Length
BLH     = TBUF+9
TAILL   = TBUF+10   ; Bytes read from last cache
TAILH   = TBUF+11
BODYSZL = TBUF+12   ; # Bytes to read at a time in Body
BODYSZH = TBUF+13
STL2    = TBUF+14   ; Payload Start address (working var)
STH2    = TBUF+15
BIN_1ST = TBUF+16   ; Flag for binary loader signature (FF -> 1st pass)

; Following for DUMP
DUMP_START = TBUF
DUMP_END   = TBUF+2

; Following for FILL
FILL_START = TBUF
FILL_END   = TBUF+2
FILL_BYTE  = TBUF+4
FILL_DIST  = TBUF+5

; Following used in DO_SAVE
INITADL = TBUF+4    ; Init Addr (lo)
INITADH = TBUF+5    ; Init Addr (hi)
RUNADL  = TBUF+6    ; Run Addr (lo)
RUNADH  = TBUF+7    ; Run Addr (hi)
COMMA_ARGS_BITFIELD = RBUF+10    ; Bit field for which addrs provided

; Used in DO_OVERLAY
SECT_CNT = TBUF

AUTORUN_QUERY_FLG   = TBUF+17   ; Flag for printing contents of autorun appkey

PGEND   = *

; ==============================================================
;         O  V  E  R  L  A  Y       R  O  U  T  I  N  E  S
;--------------------------------------------------------------
; Overlay commands reside on sectors not loaded into RAM during
; boot. When the corresponding command is executed by the user,
; the DO_OVERLAY routine will load the 1 or 2 sectors into RAM
; at OVLBUF (currently $1600) and then jump to the code there.
;
; Some addresses need to be assembled as if the code were already
; loaded into OVLBUF, so you will see some assembler math used to
; derive the address needed at runtime.
; ==============================================================

;--------------------------------------
OVL_AUTORUN:
;--------------------------------------
    ; Change URL stored in AUTORUN app key
    ;-----------------------------------
        LDA     CMDSEP          ; Check if there's any arg
        BNE     AUTORUN_NEXT1   ; If arg found, skip ahead

    ; Here if no command line arg found
    ; Print error message and exit
        LDA     #<(OVLBUF-OVL_AUTORUN+AUTORUN_ERROR_STR)
        LDY     #>(OVLBUF-OVL_AUTORUN+AUTORUN_ERROR_STR)
        JMP     PRINT_STRING

AUTORUN_NEXT1:
    ; Point to start of arg on command line
        CLC
        ADC     INBUFF          ; INBUFF += CMDSEP
        STA     INBUFF
        STA     APPKEYWRITEDCB+DCB_IDX.DBUFL

    ; If "AUTORUN ?" Then abuse AUTORUN_SUBMIT to print appkey
        LDY     #$00
        LDA     #'?'
        STA     AUTORUN_QUERY_FLG
        CMP     (INBUFF),Y
        BNE     @+
        JMP     SUBMIT_AUTORUN

    ; Open app key
@:      LDA     #$01            ; Open for write (1)
        STA     AUTORUN_QUERY_FLG
        STA     AUTORUN_APPKEY+4
        LDA     #<APPKEYOPENDCB
        LDY     #>APPKEYOPENDCB
        JSR     DOSIOV

    ; Find length of URL (arg1)
        LDY     #$FF            ; Init strlen
AUTORUN_LOOP1
```

```
        INY                     ; Incr strlen
        LDA     (INBUFF),Y
        CMP     #EOL            ; At end of string?
        BNE     AUTORUN_LOOP1   ; No. Keep counting

        LDA     #LF             ; Convert EOL to LF
        STA     (INBUFF),Y
        INY                     ; One more for strlen

AUTORUN_NEXT2:
    ; Write app key
        STY     APPKEYWRITEDCB+DCB_IDX.DAUX1    ; Y = strlen
        LDA     #<APPKEYWRITEDCB
        LDY     #>APPKEYWRITEDCB
        JSR     DOSIOV

    ; Close app key
        LDA     #<APPKEYCLOSEDCB
        LDY     #>APPKEYCLOSEDCB
        JMP     DOSIOV

AUTORUN_ERROR_STR:
        .BYTE   'PATH?',EOL

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_AUTORUN:

;--------------------------------------
OVL_BASIC:
;--------------------------------------
    ; Enable or disable BASIC (or, say, U1MB ROM)
    ; Usage: [BASIC|ROM] [ON|OFF]

    ; Quit if no internal BASIC
        JSR     OVLBUF-OVL_BASIC+CHECK_INTERNAL_BASIC
        BCS     BASIC_QUIT

    ; Check for usage. arg (BASIC ON|OFF)
        LDX     CMDSEP
        LDA     LNBUF,X
        AND     #%11011111  ; Convert lower to upper
        CMP     #'O'        ; Is 1st char O? as in ON|OFF
        BNE     BASIC_USAGE
        INX
        LDA     LNBUF,X
        AND     #%11011111  ; Convert lower to upper
        CMP     #'F'        ; Is 2nd char F? as in OFF?
        BEQ     BASIC_OFF
        CMP     #'N'        ; Is 2nd char N? as in ON?
        BNE     BASIC_USAGE

    ;--------------------------------------
    ; We are here if BASIC ON or ROM ON was the command.
    ; Do a favor and jump to CAR if ROM is already enabled
    ;--------------------------------------
        LDA     PORTB
        AND     #%00000010
        BNE     BASIC_ON
        JMP     DO_CAR

BASIC_ON:
    ;--------------------------------------
    ; Source: ANTIC Volume 4 #10 Feb 1986
    ; BASIC ON/OFF Switcher [Chadwick]

    ;--------------------------------------
    ; Enable BASIC in XL/XE
    ;--------------------------------------
        LDA     #$00
        STA     BASICF      ; BASIC RAM FLAG
        LDA     #$52
        STA     $03EB       ; CARTRIDGE CHECKSUM

        LDA     PORTB
        AND     #%11111101  ; If Bit 1 = 0 then BASIC is enabled
        STA     PORTB       ; BASIC ROM FLAG
        BNE     BASIC_WARM  ; Always jump

BASIC_OFF:
    ;--------------------------------------
    ; Disable BASIC in XL/XE
    ;--------------------------------------
        LDA     #$01
        STA     BASICF
    ;--------------------------------------
    ; Change border color as reminder that
    ; we're working with limited addr space
    ;--------------------------------------
        LDA     COLOR4_ORIG
        STA     COLOR4

BASIC_WARM:
        JMP     WARMSV  ; XL/XE WARMSTART

BASIC_QUIT:
        RTS

BASIC_USAGE:
        LDA     #<(OVLBUF-OVL_BASIC+BASIC_ERROR)
        LDY     #>(OVLBUF-OVL_BASIC+BASIC_ERROR)
        JMP     PRINT_STRING

BASIC_ERROR:
        .BYTE   '[BASIC|ROM] [ON|OFF]',EOL

;--------------------------------------
CHECK_INTERNAL_BASIC:
;--------------------------------------
    ; Check for internal BASIC found in XL/XEs
    ; On return:
    ;   CARRY is clear if not found
    ;   CARRY is set if found
    ;--------------------------------
        CLC
        LDA     $FFF7
        CMP     #$FF        ; ????
        BEQ     NOBASIC_ERROR
        CMP     #$DD        ; OSA NTSC
        BEQ     NOBASIC_ERROR
        CMP     #$F3        ; OSB NTSC
        BEQ     NOBASIC_ERROR
        CMP     #$D6        ; OSA PAL
        BEQ     NOBASIC_ERROR
        CMP     #$22        ; OSB PAL
        BEQ     NOBASIC_ERROR
        CMP     #$0A        ; OSA 1200XL
        BEQ     NOBASIC_ERROR
        CMP     #$0B        ; OSB 1200XL
        BEQ     NOBASIC_ERROR
        RTS

;--------------------------------------
NOBASIC_ERROR:
;--------------------------------------
        LDA     #<(OVLBUF-OVL_BASIC+NOBASIC_ERROR_STR)
        LDY     #>(OVLBUF-OVL_BASIC+NOBASIC_ERROR_STR)
        JSR     PRINT_STRING
        SEC
        RTS

NOBASIC_ERROR_STR:
        .BYTE   'NO BUILT-IN BASIC',EOL

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_BASIC:

;--------------------------------------
OVL_DIR:
;--------------------------------------
        JSR     OVLBUF-OVL_DIR+DIR_INIT   ; JSR DIR_INIT
        JSR     OVLBUF-OVL_DIR+DIR_OPEN   ; JSR_DIR_OPEN
        CPY     #$01        ; success (1) ?
        BEQ     DIR_LOOP    ; if success, jump ahead
        JMP     PRINT_ERROR ; exit

DIR_LOOP:
    ;--------------------------------------
    ; Send Status request to SIO
    ;--------------------------------------
        LDA     #<STADCB
        LDY     #>STADCB
        JSR     DOSIOV

    ;--------------------------------------
    ; Status returns DVSTAT
    ;--------------------------------------
        LDX     #$00
        CPX     DVSTAT+1    ; if byte count < 255 (that is, hi=0)
        BEQ     DIR_LT_255  ; then skip

    ;--------------------------------------
    ; Branch 1: Read 255 bytes (max)
    ;--------------------------------------
        DEX                 ; X now 255 (Read FF Bytes)
        STX     OVLBUF-OVL_DIR+DIRRDCB+DCB_IDX.DBYTL
        STX     OVLBUF-OVL_DIR+DIRRDCB+DCB_IDX.DAUX1
        BMI     DIR_NEXT1   ; "always" true. skip down SIO call

    ;--------------------------------------
    ; Branch 2: Read < 255 bytes
    ;--------------------------------------
DIR_LT_255:
        LDA     DVSTAT      ; Get count of bytes remaining
        BEQ     DIR_ERROR   ; If here then DVSTAT = $0000 (error)
        STA     OVLBUF-OVL_DIR+DIRRDCB+DCB_IDX.DBYTL
        STA     OVLBUF-OVL_DIR+DIRRDCB+DCB_IDX.DAUX1

    ;-------------------------
    ; Send Read request to SIO
    ;-------------------------
DIR_NEXT1:
        LDA     #<(OVLBUF-OVL_DIR+DIRRDCB)
        LDY     #>(OVLBUF-OVL_DIR+DIRRDCB)
        JSR     DOSIOV      ; Fetch directory listing
        JSR     OVLBUF-OVL_DIR+DIR_PRINT   ; JSR DIR_PRINT

    ;--------------------------------------
    ; Pause output if SPACE key code found
    ;--------------------------------------
DIR_WAIT:
        LDA     CH
        CMP     #SPC_KEY
        BEQ     DIR_WAIT

    ;--------------------------------------
    ; Exit loop if ESC key code found
    ;--------------------------------------
        LDA     CH
        CMP     #ESC_KEY    ; hardware code for ESC key
        BEQ     DIR_NEXT

    ;--------------------------------------
    ; Exit loop if Break key code pressed
    ;--------------------------------------

    ;--------------------------------------
    ; Loop if more data to read
    ;--------------------------------------
        LDA     DVSTAT+1    ; Was there more to read (that is, was hi>0)?
        BNE     DIR_LOOP    ; If yes, then do it again

DIR_NEXT:
        LDA     #$FF        ; Clear key
        STA     CH
        JMP     OVLBUF-OVL_DIR+DIR_CLOSE

DIRRDCB:
        .BYTE   DEVIDN      ; DDEVIC
        .BYTE   $FF         ; DUNIT
        .BYTE   'R'         ; DCOMND
        .BYTE   $40         ; DSTATS
        .BYTE   <RBUF       ; DBUFL
        .BYTE   >RBUF       ; DBUFH
        .BYTE   $FE         ; DTIMLO
        .BYTE   $00         ; DRESVD
        .BYTE   $00         ; DBYTL
        .BYTE   $00         ; DBYTH
        .BYTE   $00         ; DAUX1
        .BYTE   $00         ; DAUX2

;--------------------------------------
; Set DUNITs in all DCBs used by DIR
;--------------------------------------
DIR_INIT:
;--------------------------------------
        JSR     GET_DOSDR   ; On return, X <- n in Nn:
        STX     OVLBUF-OVL_DIR+DIRODCB+DCB_IDX.DUNIT   ; DUNIT for Open
        STX     STADCB+DCB_IDX.DUNIT    ; DUNIT for Status
        STX     OVLBUF-OVL_DIR+DIRRDCB+DCB_IDX.DUNIT   ; DUNIT for Read
        STX     CLODCB+DCB_IDX.DUNIT    ; DUNIT for Close
        RTS

;--------------------------------------
DIR_OPEN:
;--------------------------------------
        JSR     PREPEND_DRIVE

    ;--------------------------------
    ; Default to arg1
    ;--------------------------------
        LDX     INBUFF
        LDY     INBUFF+1
```

```
    ;-----------------------------------
    ; But use Nn:*.* if no arg1
    ;-----------------------------------
        LDA     CMDSEP          ; 0 means no arg1
        BNE     DIR_OPEN_NEXT   ; If arg1 present then skip

    ;-----------------------------------
    ; Here if no arg1
    ;-----------------------------------
        LDX     #<(OVLBUF-OVL_DIR+DIR_OPEN_STR)
        LDY     #>(OVLBUF-OVL_DIR+DIR_OPEN_STR)

        LDA     DOSDR
        ORA     #'0'            ; Convert, say, 1 to '1'
        STA     OVLBUF-OVL_DIR+DIR_OPEN_STR+1  ; Inject DOSDR into string

DIR_OPEN_NEXT:
        STX     OVLBUF-OVL_DIR+DIRODCB+DCB_IDX.DBUFL
        STY     OVLBUF-OVL_DIR+DIRODCB+DCB_IDX.DBUFH

        LDA     #<(OVLBUF-OVL_DIR+DIRODCB)
        LDY     #>(OVLBUF-OVL_DIR+DIRODCB)
        JMP     DOSIOV

;---------------------------------------
DIR_ERROR:
;---------------------------------------
        LDA     #<(OVLBUF-OVL_DIR+DIR_ERROR_STR)
        LDY     #>(OVLBUF-OVL_DIR+DIR_ERROR_STR)
        JSR     PRINT_STRING
        LDY     #$01            ; Return error
        RTS

DIR_ERROR_STR:
        .BYTE   'UNABLE TO READ DIR',EOL

DIR_OPEN_STR:
        .BYTE   'N :*.*',EOL

DIRODCB:
        .BYTE   DEVIDN          ; DDEVIC
        .BYTE   $FF             ; DUNIT
        .BYTE   'O'             ; DCOMND
        .BYTE   $80             ; DSTATS
        .BYTE   $FF             ; DBUFL
        .BYTE   $FF             ; DBUFH
        .BYTE   $FE             ; DTIMLO
        .BYTE   $00             ; DRESVD
        .BYTE   $00             ; DBYTL
        .BYTE   $01             ; DBYTH
        .BYTE   $06             ; DAUX1
        .BYTE   $80             ; DAUX2 (Long Dir)

; End of DIR_OPEN
;---------------------------------------

;---------------------------------------
DIR_PRINT:
;---------------------------------------
        ; Print results using CIO
        LDX     #$00
        LDA     #PUTCHR
        STA     ICCOM,X

        ; Fill out buffer loc
        LDA     #<RBUF
        STA     ICBAL,X
        LDA     #>RBUF
        STA     ICBAH,X

        ; Fill out size loc
        LDA     OVLBUF-OVL_DIR+DIRRDCB+DCB_IDX.DBYTL
        STA     ICBLL,X
        TXA
        STA     ICBLH,X
        JMP     CIOV

;---------------------------------------
DIR_CLOSE:
;---------------------------------------
        ; Close
        LDA     #<CLODCB
        LDY     #>CLODCB
        JMP     DOSIOV

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_DIR:

;---------------------------------------
OVL_DUMP:
;---------------------------------------
    ; Dump a range of memory to the screen

    ; Convert comma-delimiters to offsets
        JSR     PARSE_COMMAS

    ; Quit if no args
        LDA     CMDSEP      ; Will contain zero if no arg 1
        BNE     DUMP_NEXT0  ; Skip ahead if args are present

    ; Otherwise show usage
        LDA     #<(OVLBUF-OVL_DUMP+DUMP_ERROR_STR)
        LDY     #>(OVLBUF-OVL_DUMP+DUMP_ERROR_STR)
        JMP     PRINT_STRING

DUMP_NEXT0:
    ; Convert arg1 (start address) from hex string to word
        TAY                 ; Y will hold offset to hex string
        CLC
        ADC     #$04        ; Expected length of hex string
        STA     RBUF        ; RBUF is used by ASCII2ADDR
        JSR     ASCII2ADDR
        BCC     @+
        RTS                 ; Quit if bad ascii 2 addr conversion

    ; Save start address as variable
    ; Also, initialize end address = start address + 7
    ; (I suspect I have an off-by-one error here or at the
    ; bound check, but it's working the way I want as it is).
@:      LDA     INBUFF
        STA     DUMP_START
        CLC
        ADC     #$07
        STA     DUMP_END
        LDA     INBUFF+1
        STA     DUMP_START+1
        STA     DUMP_END+1
        BCC     @+
        INC     DUMP_END+1

    ; Check for arg 2
@:      LDY     CMDSEP+1    ; Offset needed in Y for ASCII2ADDR
        BEQ     DUMP_NEXT2  ; Skip ahead if arg 2 (end address) is not given

DUMP_NEXT1:
    ; Arg 2 given, convert string to address
        ; Convert hex string to int
        TYA                 ; Y = CMDSEP+1 - needed for ASCII2ADDR
        CLC                 ; Offset + 4 needed needed for ASCII2ADDR
        ADC     #$04
        STA     RBUF
        JSR     ASCII2ADDR  ; LNBUF,Y contains 4-char addr
        BCC     @+
        RTS                 ; Quit if bad ascii 2 addr conv

@:      LDA     INBUFF
        STA     DUMP_END
        LDA     INBUFF+1
        STA     DUMP_END+1

DUMP_NEXT2:
    ; Initalize index into memory being dumped
        LDY     #$00

    ; Get address
        LDA     DUMP_START
        STA     INBUFF
        PHA                                 ; Push DUMP_START (Lo byte)

        LDA     DUMP_START+1
        STA     INBUFF+1

    ; Print address
        JSR     OVLBUF-OVL_DUMP+INT2ASCII   ; Print Hi byte. Re-use X = 0

        PLA                                 ; Pull DUMP_START (Lo byte)
        JSR     OVLBUF-OVL_DUMP+INT2ASCII   ; Print Lo byte

    ; Loop for 8 bytes
DUMP_LOOP0:
        LDA     #' '            ; Put a space between hex bytes
        STA     RBUF,X
        INX

    ; Fetch byte from memory
        LDA     (INBUFF),Y
    ; and convert/put hex string into RBUF
        JSR     OVLBUF-OVL_DUMP+INT2ASCII

    ; Advance to next position
    ; Loop until 8 bytes displayed
        INY
        CPX     #$1A
        BCC     DUMP_LOOP0

    ; Here if done fetching, converting, displaying 8 bytes
        LDA     #EOL
        STA     RBUF,X

    ; Print the whole line of addr + bytes
        LDA     #<RBUF
        LDY     #>RBUF
        JSR     PRINT_STRING

    ; Increment address
        CLC
        LDA     #$08
        ADC     DUMP_START      ; Start += 8
        STA     DUMP_START
        BCC     @+

        CLC
        LDA     #$01            ; Use ADC to catch carry
        ADC     DUMP_START+1
        STA     DUMP_START+1
        BCS     DUMP_DONE       ; Quit if we went from FFxx to 00xx

    ; Are we done yet? Is START <= END?
@:      LDA     DUMP_END
        CMP     DUMP_START
        LDA     DUMP_END+1
        SBC     DUMP_START+1
        BCS     @+

DUMP_DONE:
        RTS

@:  ; Check for keypress
        LDA     CH              ; Will be $FF if no keypress
        CMP     #ESC_KEY        ; Leave if ESC key pressed
        BEQ     DUMP_DONE
    ; Process next line
        JMP     OVLBUF-OVL_DUMP+DUMP_NEXT2

;--------------
INT2ASCII:
;--------------
    ; Input A contains byte, X contains offset to output string
    ; Output RBUF+X contains string
    ; Stolen from WOZMON disassembly by Jeff Tranter
        PHA                 ; Save A for LSD
        LSR
        LSR
        LSR
        LSR                 ; MSD to LSD position.
        JSR     OVLBUF-OVL_DUMP+PRHEX
        PLA
PRHEX:  AND     #$0F        ; Mask LSD for hex print
        ORA     #'0'        ; Add "0".
        CMP     #$3A        ; Digit?
        BCC     ECHO        ; Yes, skip ahead
        ADC     #$06        ; Add offset for letter
ECHO:   STA     RBUF,X
        INX
        RTS

DUMP_ERROR_STR:
        .BYTE   'DUMP START [END]',EOL

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_DUMP:

;---------------------------------------
OVL_FILL:
;---------------------------------------
    ; Fill a range of memory with a given byte

    ; Convert comma-delimiters to offsets
        JSR     PARSE_COMMAS

    ; Quit if 3 args not given
```

```
        LDA     CMDSEP      ; Will contain zero if no arg 1
        BNE     FILL_NEXT0

    ; Otherwise show usage
FILL_ERROR:
        LDA     #<(OVLBUF-OVL_FILL+FILL_ERROR_STR)
        LDY     #>(OVLBUF-OVL_FILL+FILL_ERROR_STR)
        JMP     PRINT_STRING

FILL_NEXT0:
    ; Convert arg1 (start address) from hex string to word
        JSR     OVLBUF-OVL_FILL+FILL_ASCII2ADDR
        BCS     FILL_HOP0

    ; Save start address as variable
        LDA     INBUFF
        STA     FILL_START
        LDA     INBUFF+1
        STA     FILL_START+1

        LDA     CMDSEP+1
        BEQ     FILL_ERROR

    ; Convert arg2 (end address) from hex string to word
        JSR     OVLBUF-OVL_FILL+FILL_ASCII2ADDR
FILL_HOP0:
        BCS     FILL_QUIT

    ; Save end address as variable
        LDA     INBUFF
        STA     FILL_END
        LDA     INBUFF+1
        STA     FILL_END+1

    ; Convert arg3 (fill byte) from hex string to byte
        LDY     CMDSEP+2
        BEQ     FILL_ERROR

        LDA     LNBUF+2,Y
        CMP     #EOL
        BNE     FILL_ERROR

        LDA     #'0'
        STA     LNBUF+2,Y
        STA     LNBUF+3,Y
        LDA     #EOL
        STA     LNBUF+4,Y
        TYA
        JSR     OVLBUF-OVL_FILL+FILL_ASCII2ADDR
        BCS     FILL_ERROR

    ; Save fill byte
        LDA     INBUFF
        STA     FILL_BYTE
        LDA     INBUFF+1
        STA     FILL_BYTE

    ; Find distance between start and end addresses
    ; fill_dist = end - start + 1
        SEC
        LDA     FILL_END
        SBC     FILL_START
        STA     FILL_DIST
        LDA     FILL_END+1
        SBC     FILL_START+1
        STA     FILL_DIST+1

    ; Quit if fill_end < fill_start
        BMI     FILL_ERROR

    ; Ready to start filling
    ; Initialize to start address
        LDA     FILL_START
        STA     INBUFF
        LDA     FILL_START+1
        STA     INBUFF+1

    ; Loop to fill
@:      LDA     FILL_BYTE
        LDY     #$00
FILL_LOOP01:
        LDA     FILL_BYTE
        STA     (INBUFF),Y

    ; Exit loop if inbuff >= fill_end
        LDA     INBUFF
        CMP     FILL_END
        LDA     INBUFF+1
        SBC     FILL_END+1
        BCS     FILL_QUIT

    ; Increment address
        CLC
        LDA     #$01
        ADC     INBUFF
        STA     INBUFF
        BCC     @+
        INC     INBUFF+1

    ; Do again
@:      JMP     OVLBUF-OVL_FILL+FILL_LOOP01

FILL_QUIT:
        RTS

FILL_ASCII2ADDR:
        TAY
        CLC
        ADC     #$04
        STA     RBUF
        JSR     ASCII2ADDR
        RTS

FILL_ERROR_STR:
        .BYTE   'FILL START END XX',EOL

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_FILL:
;--------------------------------------
OVL_HELP:
;--------------------------------------
    ; Append either "HELP" or arg1 to URL
        LDX     #$00        ; index to start of article buf
        LDY     CMDSEP      ; index to cmd line arg

HELP_LOOP1:
        LDA     (INBUFF),Y
        CMP     #EOL
        BEQ     HELP_NEXT1  ; Exit loop at end of arg
        CPX     #22
        BPL     HELP_DONE   ; Exit if arg is too long

    ; Convert lower-case to upper-case
        JSR     TOUPPER
        STA     OVLBUF-OVL_HELP+HELP_ARTICLE,X
        INX
        INY
        BNE     HELP_LOOP1  ; Always true

    ; Append .DOC extension to article name
HELP_EXT:
        .BYTE   '.DOC',EOL,$00

HELP_NEXT1:
        LDY     #$00

HELP_LOOP2:
        LDA     OVLBUF-OVL_HELP+HELP_EXT,Y
        STA     OVLBUF-OVL_HELP+HELP_ARTICLE,X  ; Store null term too
        BEQ     HELP_NEXT2      ; Skip ahead if terminator
        INX
        INY
        BNE     HELP_LOOP2  ; Always true

HELP_NEXT2:
    ; Copy URL to LNBUF
        LDX     #$00    ; Index to start of HELP_URL
        LDY     #$05    ; Index to start at arg1 for "TYPE "

HELP_LOOP3:
        LDA     OVLBUF-OVL_HELP+HELP_URL,X  ; Get source byte
        STA     LNBUF,Y     ; Write to target location
        BEQ     HELP_DONE   ; Exit loop on null terminator
        INX                 ; Advance indices
        INY
        BNE     HELP_LOOP3  ; Always true

HELP_DONE:
        LDA     #$05        ; Trick TYPE to look for URL in arg1
        STA     CMDSEP
        JMP     DO_TYPE

HELP_URL:
        .BYTE   'N4:HTTPS://raw.githubusercontent.com/michaelsternberg/fujinet-nhandler/nos/
nos/HELP/'

HELP_ARTICLE:
    :24 .BYTE   $00

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_HELP:

;--------------------------------------
OVL_NCOPY:
;--------------------------------------
    ; Check if an arg exists
        LDY     CMDSEP          ; Check if there's any args
        BEQ     NCOPY_ERROR     ; No. Show usage and quit

    ; Default to Open with Truncate
    ; Later, if we find 'A' as 3rd arg
    ; Then we'll change the mode to Append
        LDA     #$08            ; Open with Truncate mode
        STA     WRITEMODE

    ; Find comma, convert comma to EOL.  Quoted names (with spaces)
    ; are already de-quoted by GETCMDTEST's QSTRIP pass, so FROM,TO
    ; is a clean contiguous string here.
        JSR     PARSE_COMMAS

    ; One-arg COPY?  With no TO given (CMDSEP+1 == 0) the destination
    ; is the currently selected drive.  Synthesize "Nn:" (n = DOSDR)
    ; just past FROM's EOL in LNBUF and point CMDSEP+1 at it.  Both the
    ; wildcard driver and the single-file path read TO from CMDSEP+1,
    ; and a bare "Nn:" target makes each copy keep the source's name.
        LDA     CMDSEP+1
        BNE     NCW_HAVETO
        LDY     CMDSEP          ; walk FROM to its EOL
NCW_DEFEOL:
        LDA     (INBUFF),Y
        CMP     #EOL
        BEQ     NCW_DEFBLD
        INY
        BNE     NCW_DEFEOL
NCW_DEFBLD:
        INY                     ; TO begins just past FROM's EOL
        STY     CMDSEP+1
        LDA     #'N'
        STA     (INBUFF),Y
        INY
        LDA     DOSDR           ; current drive digit
        ORA     #'0'
        STA     (INBUFF),Y
        INY
        LDA     #':'
        STA     (INBUFF),Y
        INY
        LDA     #EOL
        STA     (INBUFF),Y
NCW_HAVETO:

    ; Wildcard FROM? -> load & run the (transient) wildcard-copy module.
    ; Scan inline so only the loader (WILD_LAUNCH) stays resident.
        LDY     CMDSEP
NCW_SCAN:
        LDA     LNBUF,Y
        CMP     #EOL
        BEQ     NCW_NONE            ; no wildcard -> normal single-file copy
        CMP     #'*'
        BEQ     NCW_WILD
        CMP     #'?'
        BEQ     NCW_WILD
        INY
        BNE     NCW_SCAN
NCW_WILD:
        LDA     #$00            ; select COPY driver
        STA     WILD_MODE
        JMP     WILD_LAUNCH
NCW_NONE:

    ; Check if 3rd arg exists
        LDY     CMDSEP+2
        BEQ     NCOPY_ARG2

    ; Here if 3rd arg found
    ; Is it an A (for APPEND option)?
        LDA     (INBUFF),Y
        JSR     TOUPPER
        CMP     #'A'
        BNE     NCOPY_ARG2
        INY
        LDA     (INBUFF),Y
```

```
        CMP     #EOL
        BNE     NCOPY_ARG2
        LDA     #$09            ; Open with Append mode
        STA     WRITEMODE

NCOPY_ARG2:
    ; Check if 2nd arg exists
        LDX     #$02            ; Used later for counting colon test passes
        LDY     CMDSEP+1        ; Check if there's any args
        BEQ     NCOPY_ERROR     ; No. Show usage and quit
        LDA     (INBUFF),Y
        CMP     #EOL            ; Check if arg2 is empty
        BNE     NCOPY_NEXT0

NCOPY_ERROR:
        LDA     #<(OVLBUF-OVL_NCOPY+NCOPY_ERROR_STR)
        LDY     #>(OVLBUF-OVL_NCOPY+NCOPY_ERROR_STR)
        JMP     PRINT_STRING

NCOPY_ERROR_STR:
       .BYTE   'NCOPY FROM,TO[,A]',EOL

NCOPY_NEXT0:
    ; Check if 2nd arg ends in '/'
    ; First find end of 2nd arg
        DEY
NCOPY_LOOP:
        INY
        LDA     (INBUFF),Y
        CMP     #EOL
        BNE     NCOPY_LOOP

    ; Check if prev char is '/'
        DEY
        LDA     (INBUFF),Y
        CMP     #'/'
        BNE     NCOPY_IS_NEOL
        INY
        BNE     NCOPY_JUMP

NCOPY_IS_NEOL:
    ; Check if 2nd arg is in the form of N:<EOL> or Nn:<EOL>
        LDY     CMDSEP+1
        LDA     (INBUFF),Y      ; First, does arg start with "N"?
        CMP     #'N'            ;
        BNE     NCOPY_NEXT_A    ; No. Give up and skip ahead.

    ; Is next char :?
NCOPY_IS_COLON:
        INY                     ; Advance to next char in arg
        LDA     (INBUFF),Y      ;
        CMP     #':'            ;
        BEQ     NCOPY_NEXT01    ; Yes, it's a ":", skip ahead

        DEX                     ; Test this only 2 times
        BEQ     NCOPY_NEXT_A    ; No. Give up and skip ahead

    ; Is char digit (0-8)?  Drives are N1..N8 (see README/DO_DRIVE_CHG).
        CMP     #'0'
        BCC     NCOPY_NEXT_A
        CMP     #'9'
        BCS     NCOPY_NEXT_A

    ; Here if Nn. Jump back to check if next char is ':'
        BCC     NCOPY_IS_COLON

    ; Here if Nn: or N:
    ; Is next char EOL?
NCOPY_NEXT01:
        INY                     ; Advance to next char in arg
        LDA     (INBUFF),Y      ;
        CMP     #EOL            ; Is the arg just Nn:<EOL> / N:<EOL>?
        BNE     NCOPY_NEXT_A    ; No. Give up and skip ahead.

    ; Here if Nn:<EOL> or N:<EOL>
    ; Copy source filename to target
    ; So, say, N1:<EOL> becomes N1:MYFILE<EOL>
NCOPY_JUMP:
        STY     OVLBUF-OVL_NCOPY+NCOPY_POS       ; Current target file offset
        LDX     CMDSEP          ; Get offset to beginning of filename

NCOPY_LOOP0:
        LDA     $0582,X
        CMP     #'/'            ; if '/' or
        BEQ     NCOPY_RESET_OFFSET
        CMP     #':'            ; if ':' then erase path so far
        BNE     NCOPY_CONTINUE
NCOPY_RESET_OFFSET:
        LDY     OVLBUF-OVL_NCOPY+NCOPY_POS
;        DEY
    BNE     @+
NCOPY_CONTINUE:
        STA     (INBUFF),Y
        INY
@:
        INX
        CMP     #EOL
        BNE     NCOPY_LOOP0

NCOPY_NEXT_A:
        CLC
        LDA     CMDSEP
        ADC     INBUFF
        STA     INBUFF
        BCC     NCOPY_NEXT_B
        INC     INBUFF+1

    ; Prepend Drive to source file (if necessary)
NCOPY_NEXT_B:
        JSR     PREPEND_DRIVE
        JMP     DO_NCOPY1           ; Load next portion of code and run it

NCOPY_POS:
        .BYTE   $00

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_NCOPY:

;--------------------------------------
OVL_NCOPY1:
;--------------------------------------
    ; Save copy source to TBUF.
    ; Needed for string comparison later between source and target
    ; Parsing target may clobber part of source string
    ; where it currently is.
        LDY     #$FF
NCOPY_LOOP1:
        INY
        LDA     (INBUFF),Y
        STA     TBUF,Y
        CMP     #EOL
        BNE     NCOPY_LOOP1

;    ; Check if Channel 1 should be closed prior to opening
;        LDX     #$10            ; Channel 1
;        LDA     IOCB,X          ; Will be $FF if closed
;        BMI     NCOPY_NEXT1     ; Skip if already closed
;        JSR     CIOCLOSE

    ; Open source file for read
NCOPY_NEXT1:
;        LDX     #$10            ; Channel 1
        JSR     NEXT_IOCB
        BCC     @+
        RTS

@:      STX     SOURCE_IOCB
        LDY     #$04            ; Open for read
        JSR     CIOOPEN
        CPY     #$80
        BCC     NCOPY_NEXT2
        JSR     PRINT_ERROR
        JMP     OVLBUF-OVL_NCOPY1+NCOPY_CLOSE

NCOPY_NEXT2:
    ; Change INBUFF to point to target filename
        CLC
        LDA     #<LNBUF         ; Low byte of line buffer
        ADC     CMDSEP+1        ; Offset to start of arg 2
        STA     INBUFF          ; Store it in pointer address
        BCC     NCOPY_NEXT3A
        INC     INBUFF+1

    ; Prepend Drive to target file
NCOPY_NEXT3A:
    ; Do no prepend drive if another device is specified
    ; such as P:, P1:, E:
        LDY     #$00
        LDA     (INBUFF),Y
        CMP     #'N'
        BEQ     NCOPY_NEXT3B    ; If 1st char is N, proceed, stop checking
        INY
        LDA     (INBUFF),Y      ; If 2nd char is :, skip filename processing
        CMP     #':'
        BEQ     NCOPY_OPEN_20
        INY
        LDA     (INBUFF),Y
        CMP     #':'            ; If 3rd char is :, skip filename procssing
        BEQ     NCOPY_OPEN_20

NCOPY_NEXT3B:
        JSR     PREPEND_DRIVE
        LDY     #$00            ; Check if N: device
        LDA     (INBUFF),Y
        CMP     #'N'
        BNE     NCOPY_OPEN_20   ; No? Skip ahead.

    ; Compare source and target. Quit if the same.
    ; Note: this is a very limited comparison. If
    ; the source and target have different paths
    ; but point to the same file then the source will
    ; get clobbered when the target is opened for write.
    ; A work-around would to open the target for read and
    ; fail if successful, that is, the target already
    ; exists when it shouldn't. An exception would be needed
    ; for append mode.
        LDY     #$FF
NCOPY_LOOP2:
        INY
        LDA     (INBUFF),Y
        CMP     TBUF,Y
        BNE     NCOPY_NEXT4
        CMP     #EOL
        BEQ     NCOPY_SAME_ERR
        BNE     NCOPY_LOOP2

    ; Source and target are the same file. Quit
NCOPY_SAME_ERR:
        LDA     #<(OVLBUF-OVL_NCOPY1+NCOPY_SAME_TXT)
        LDY     #>(OVLBUF-OVL_NCOPY1+NCOPY_SAME_TXT)
        JSR     PRINT_STRING
        JMP     OVLBUF-OVL_NCOPY1+NCOPY_CLOSE

NCOPY_NEXT4:
    ; Get NPWD of target
        LDY     #$01
        LDA     (INBUFF),Y      ; Get n in Nn: for target
        AND     #$0F            ; Convert '2' to 2
        TAX                     ; X needs to have DUNIT
        JSR     NPWD_ENTRY

    ; Find location of EOL for NPWD
        LDX     #$FF
NCOPY_LOOP3:
        INX
        LDA     RBUF,X
        CMP     #EOL
        BNE     NCOPY_LOOP3

    ; Copy target file to tail of NPWD
        DEX             ; Step back 1 char pos
        LDY     #$02    ; Point to 1 char before filename
NCOPY_LOOP4:
        INY
        INX
        LDA     (INBUFF),Y
        STA     RBUF,X
        CMP     #EOL
        BNE     NCOPY_LOOP4

    ; Copy N4: to the start of the path
        LDY     #$02
NCOPY_LOOP5:
        LDA     OVLBUF-OVL_NCOPY1+NCOPY_N8,Y
        STA     RBUF-3,Y
        DEY
        BPL     NCOPY_LOOP5

    ; Use new path found in RBUF for filename
        LDA     #<(RBUF-3)
        STA     INBUFF
        LDA     #>(RBUF-3)
        STA     INBUFF+1

NCOPY_OPEN_20:
    ; Open target file
        JSR     NEXT_IOCB   ; On return, X will be like $20
        BCS     NCOPY_QUIT1 ; On failure, close source IOCB and quit
        STX     TARGET_IOCB
        LDY     WRITEMODE   ; 8=truncate 9=append
        JSR     CIOOPEN
        CPY     #$80        ; Check for error
```

```
        BCC     NCOPY_CONT  ; Jump ahead if no error
        JSR     PRINT_ERROR
        JMP     OVLBUF-OVL_NCOPY1+NCOPY_CLOSE   ; Close IOCBs and exit

    ; We've run out of space at OVLBUF
    ; Process the next part as another
    ; overlay command.

NCOPY_CONT:
        LDA     #$FF        ; Clear prev overlay id
        STA     OVLPRV      ; to avoid caching confusion
        JMP     DO_NCOPY2

    ; Close routine
NCOPY_CLOSE:
        LDX     TARGET_IOCB
        JSR     CIOCLOSE
NCOPY_QUIT1:
        LDX     SOURCE_IOCB
        JSR     CIOCLOSE
NCOPY_QUIT2:
        RTS


NCOPY_SAME_TXT:
        .BYTE   'SAME FILE?',EOL

NCOPY_N8:
        .BYTE   'N4:'

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_NCOPY1:

;--------------------------------------
OVL_NCOPY2:
;--------------------------------------
    ; Copy the file body in big bursts.  Point the CIO
    ; buffer at free RAM above DOS (MEMLO) and move up to
    ; MAXBURST bytes per pass: a binary GET/PUT BYTES with a
    ; buffer >= MINBURST makes the N: handler burst the whole
    ; block in one SIO frame straight to/from this buffer,
    ; instead of trickling 128 bytes at a time.  MEMLO is free
    ; RAM above DOS, well clear of the overlay and the screen.
        LDA     MEMLO
        STA     INBUFF
        LDA     MEMLO+1
        STA     INBUFF+1

NCOPY2_NEXT2:
        LDA     #$00                ; assume more blocks follow
        STA     OVLBUF-OVL_NCOPY2+NCOPY2_LOOP_FLG

    ; Read up to MAXBURST bytes from the source.
        LDX     SOURCE_IOCB         ; X will be like $10
        LDA     #<MAXBURST
        LDY     #>MAXBURST
        JSR     CIOGET
        BPL     NCOPY2_WRITE        ; success -> full block read

    ; Here if error/EOF status. If not EOF, print and bail.
        CPY     #136                ; EOF?
        BEQ     NCOPY2_EOF_FOUND    ; Yes, this is the final block
        JSR     PRINT_ERROR
        JMP     OVLBUF-OVL_NCOPY2+NCOPY2_CLOSE

NCOPY2_EOF_FOUND:
        LDA     #$01                ; mark this as the last block
        STA     OVLBUF-OVL_NCOPY2+NCOPY2_LOOP_FLG

    ; Write exactly the number of bytes CIO transferred.
    ; ICBLL/ICBLH is the actual byte count (= MAXBURST on a
    ; full read, a short count on the final block).
NCOPY2_WRITE:
        LDX     SOURCE_IOCB
        LDA     ICBLL,X
        STA     OVLBUF-OVL_NCOPY2+NCOPY2_LEN
        LDA     ICBLH,X
        STA     OVLBUF-OVL_NCOPY2+NCOPY2_LEN+1
        ORA     OVLBUF-OVL_NCOPY2+NCOPY2_LEN
        BEQ     NCOPY2_ENDCHK       ; nothing read -> skip the write

        LDX     TARGET_IOCB
        LDA     OVLBUF-OVL_NCOPY2+NCOPY2_LEN
        LDY     OVLBUF-OVL_NCOPY2+NCOPY2_LEN+1
        JSR     CIOPUT

NCOPY2_ENDCHK:
        LDA     OVLBUF-OVL_NCOPY2+NCOPY2_LOOP_FLG
        BEQ     NCOPY2_NEXT2        ; not EOF -> next block

NCOPY2_CLOSE:
        LDX     SOURCE_IOCB         ; X will be like $10
        JSR     CIOCLOSE
        LDX     TARGET_IOCB         ; X will be like $20
        JMP     CIOCLOSE

NCOPY2_LOOP_FLG:
        .BYTE  $00
NCOPY2_LEN:
        .BYTE  $00,$00

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_NCOPY2:

;--------------------------------------
OVL_NDEL:
;--------------------------------------
    ; DEL/ERA scan overlay.  Look for a wildcard ('*'/'?') anywhere in
    ; the file spec.  If present, run the wildcard (Y/N) delete driver;
    ; otherwise fall through to the ordinary single-file delete path.
    ; Position-independent: only absolute refs (LNBUF/CMDSEP/WILD_MODE)
    ; and JMPs to resident labels, so it runs correctly from OVLBUF.
        LDY     CMDSEP              ; offset to arg1 in LNBUF
        BEQ     OND_SINGLE          ; no arg -> keep current DO_GENERIC behavior
OND_SCAN:
        LDA     LNBUF,Y
        CMP     #EOL
        BEQ     OND_SINGLE          ; no wildcard -> plain single-file delete
        CMP     #'*'
        BEQ     OND_WILD
        CMP     #'?'
        BEQ     OND_WILD
        INY
        BNE     OND_SCAN
OND_WILD:
        LDA     #$01                ; select DEL driver
        STA     WILD_MODE
        JMP     WILD_LAUNCH
OND_SINGLE:
        LDX     #CMD_IDX.DEL        ; CMD_DCOMND index for DO_GENERIC
        JMP     DO_GENERIC          ; unchanged single-delete + remount

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_NDEL:

;--------------------------------------
OVL_NTRANS:
;--------------------------------------
    ; Tell the FujiNet how to translate (or not) end-of-line
    ; characters between the ATARI and the remote computer
    ;--------------------------------------
        LDX     CMDSEP          ; Check if there's any args
        BEQ     NTRANS_ERROR    ; No. Show usage and quit

        LDA     DOSDR           ; Go with current drive for now
        ;STA     OVLBUF-OVL_NTRANS+NTRDCB+DCB_IDX.DUNIT    ; it'll be overwritten later if
req'd
        STA     NTRDCB+DCB_IDX.DUNIT    ; it'll be overwritten later if req'd

    ;--------------------------------------
    ; Check for argc = 2
    ;--------------------------------------
        LDY     CMDSEP          ; Stash offset to arg1 in Y
        LDX     CMDSEP+1        ; Is there an arg2?
        BEQ     PARSE_MODE      ; No. parse arg1 as mode (0-3)

    ;--------------------------------------
    ; Here if argc = 2 (arg1 = Nn: arg2 = mode)
    ;--------------------------------------
        LDX     CMDSEP          ; Get offset to arg1 (Nn:)
        LDA     LNBUF,X         ; Is arg1's (N[n]:) 1st char = 'N'?
        CMP     #'N'            ;
        BNE     NTRANS_ERROR    ; No? Then quit
        LDA     LNBUF+1,X
        CMP     #':'            ; Is arg1's (N[n]:) 2nd char = ':'?
        BEQ     PARSE_MODE      ; Yes, stick with default drive

    ;--------------------------------------
    ; Parse drive number
    ;--------------------------------------
        CMP     #'1'            ; Quit if n in Nn not 1..4
        BCC     NTRANS_ERROR    ; Quit if < '1'
        CMP     #'9'
        BCS     NTRANS_ERROR    ; Quit if >= '9'
        EOR     #%00110000
        ;STA     OVLBUF-OVL_NTRANS+NTRDCB+DCB_IDX.DUNIT
        STA     NTRDCB+DCB_IDX.DUNIT
        LDY     CMDSEP+1

    ;--------------------------------------
    ; Confirm valid parameter
    ;--------------------------------------
PARSE_MODE:
        LDA     LNBUF,Y         ; Quit if mode not 0..3
        CMP     #'0'
        BCC     NTRANS_ERROR
        CMP     #'4'
        BCS     NTRANS_ERROR
        EOR     #%00110000      ; Here if valid parameter
        ;STA     OVLBUF-OVL_NTRANS+NTRDCB+DCB_IDX.DAUX2    ; Assign parameter to DCB
        STA     NTRDCB+DCB_IDX.DAUX2    ; Assign parameter to DCB

    ;--------------------------------------
    ; Call SIO
    ;--------------------------------------
NTRANS_CALL:
        LDA     #<NTRDCB        ; DCB can't be in Overlay
        LDY     #>NTRDCB        ; as it is used by DO_LOAD
        JSR     DOSIOV
        JMP     PRINT_ERROR

NTRANS_ERROR:
        LDA     #<(OVLBUF-OVL_NTRANS+NTRANS_ERROR_STR)
        LDY     #>(OVLBUF-OVL_NTRANS+NTRANS_ERROR_STR)
        JMP     PRINT_STRING

NTRANS_ERROR_STR:
        .BYTE   'MODE? 0=NONE, 1=CR, 2=LF, 3=CR/LF',EOL

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_NTRANS:

;--------------------------------------
OVL_REENTER:
;--------------------------------------
    ; Jump to the address stored in RUNAD or INITAD
    ; Do the one that isn't pointing to R (RUNAD first)

    ; Skip it all if both contain $0000
        LDA     INITAD
        BNE     DO_REENTER_CONT
        LDA     INITAD+1
        BNE     DO_REENTER_CONT
        LDA     RUNAD
        BNE     DO_REENTER_CONT
        LDA     RUNAD+1
        BNE     DO_REENTER_CONT

        LDA     #<(OVLBUF-OVL_REENTER+DO_REENTER_ERR)
        LDY     #>(OVLBUF-OVL_REENTER+DO_REENTER_ERR)
        JMP     PRINT_STRING

DO_REENTER_CONT:
        LDA     RUNAD
        CMP     #>R
        BNE     DO_REENTER_RUNAD
        LDA     RUNAD+1
        CMP     #>R
        BEQ     DO_REENTER_INITAD

DO_REENTER_RUNAD:
        JMP     (RUNAD)     ; Godspeed

DO_REENTER_INITAD:
        JMP     (INITAD)    ; Godspeed

DO_REENTER_ERR:
        .BYTE   'NO ADDR IN INITAD OR RUNAD',EOL

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_REENTER:

;--------------------------------------
OVL_SAVE:
;--------------------------------------
    ; INBUFF points to Filename
    ; LNBUF,Y is start of 4 char ASCII hex string
    ;--------------------------------------

    ;--------------------------------------------
    ; Convert commas to EOLs & store offsets in CMDSEP[I]
    ;--------------------------------------------
```

```
        JSR     PARSE_COMMAS
    ;---------------------------------------------
    ; Convert hex strings in arg list to lo/hi pairs
    ;---------------------------------------------
        LDX     #$00            ; Index to TBUF,X for output
        LDY     #$01            ; Index to CMDSEP for list of addr strings
    ;---------------------------------------------
    ; Prep 'ascii to addr' subroutine inputs
    ;---------------------------------------------
SAVE_LOOP:
        LDA     SAVE_ADDR_FLG,Y
        AND     COMMA_ARGS_BITFIELD
        BEQ     SAVE_SKIP       ; If arg is null then skip
        TYA
        PHA
        LDA     CMDSEP,Y
        TAY                     ; Y -> offset into CMDSEP
        CLC
        ADC     #$04            ; offset to end of arg
        STA     RBUF
    ;---------------------------------------------
    ; Call 'ascii to addr' subroutine
    ;---------------------------------------------
        TXA
        PHA
        JSR     ASCII2ADDR      ; Y contains offset into LNBUF
        PLA
        TAX
        PLA
        TAY
        BCS     SAVE_USAGE      ; Quit if invalid addr
    ;---------------------------------------------
    ; Save conversion function outputs (lo/hi)
    ;---------------------------------------------
        LDA     INBUFF          ; Get lo byte
        STA     STL,X           ; Store output of ASCII2ADDR (STL = Start Addr Lo)
        LDA     INBUFF+1        ; Get hi byte
        STA     STL+1,X         ; Store output of ASCII2ADDR
SAVE_SKIP:
        INX
        INX
        INY
        CPY     #$05
        BNE     SAVE_LOOP

    ;---------------------------------------------
    ; Quit if required filename, start, end addr not provided
    ;---------------------------------------------
        LDA     #%00000111      ; bit 0->filename bit 1->start bit 2->end
        AND     COMMA_ARGS_BITFIELD
        CMP     #%00000111      ; Quit if not all 3 provided
        BEQ     SAVE_SKIP2

SAVE_USAGE:
        LDA     #<SAVE_ERROR_STR
        LDY     #>SAVE_ERROR_STR
        JMP     PRINT_STRING

SAVE_SKIP2:
    ; Calc header (Result in BLL,BLH)
        JSR     LOAD_BUFLEN

        LDX     #$04
@:      LDA     STL,X
        STA     SAVE_HEADER+2,X
        DEX
        BPL     @-

        LDA     BLL
        STA     SAVE_HEADER+6
        LDA     BLH
        STA     SAVE_HEADER+7

   ; Save initad (this may contain garbage but will be skipped if unnecessary)
        LDA     INITADL
        STA     SAVE_INIT+4
        LDA     INITADH
        STA     SAVE_INIT+5

    ; Save runad (this may contain garbage but will be skipped if unnecessary)
        LDA     RUNADL
        STA     SAVE_RUN+4
        LDA     RUNADH
        STA     SAVE_RUN+5

    ;---------------------------------------------
    ; Get filename
    ;---------------------------------------------
        JSR     GET_DOSDR
        JSR     PREPEND_DRIVE

;    ; Open file for write
;       LDX     #$10
;       JSR     CIOCLOSE

    ; Inbuff already contains filename
;       LDX     #$10
        JSR     NEXT_IOCB       ; Get next available IOCB channel
        BCC     @+              ; Carry set if no available channels
        RTS                     ; Quit
@:      STX     TARGET_IOCB     ; X will be like $100
        LDY     #OOUTPUT
        JSR     CIOOPEN

    ; Save binary file header
;       LDX     #$10
        LDX     TARGET_IOCB
        LDA     #<SAVE_HEADER
        STA     INBUFF
        LDA     #>SAVE_HEADER
        STA     INBUFF+1
        LDA     #$06            ; A = ICBLL
        LDY     #$00            ; Y = ICBLH
        JSR     CIOPUT
        JSR     PRINT_ERROR

    ; Save binary payload
        ;LDX     #$10
        LDX     TARGET_IOCB
        LDA     SAVE_HEADER+2   ; Start Address (Lo)
        STA     INBUFF
        LDA     SAVE_HEADER+3   ; Start Address (Hi)
        STA     INBUFF+1
        LDA     SAVE_HEADER+6   ; BLL
        LDY     SAVE_HEADER+7   ; BLH
        JSR     CIOPUT
        JSR     PRINT_ERROR

    ; Optionally save init address block
        LDA     #%00001000
        AND     COMMA_ARGS_BITFIELD
        BEQ     DO_SAVE_RUNAD

        ;LDX     #$10
        LDX     TARGET_IOCB
        LDA     #<SAVE_INIT
        STA     INBUFF
        LDA     #>SAVE_INIT
        STA     INBUFF+1
        LDA     #$06        ; Writing 6 bytes
        LDY     #$00
        JSR     CIOPUT
        JSR     PRINT_ERROR

   ; Optionally save run address block
DO_SAVE_RUNAD:
        LDA     #%00010000
        AND     COMMA_ARGS_BITFIELD
        BEQ     DO_SAVE_QUIT

        ;LDX     #$10
        LDX     TARGET_IOCB
        LDA     #<SAVE_RUN
        STA     INBUFF
        LDA     #>SAVE_RUN
        STA     INBUFF+1
        LDA     #$06        ; Writing 6 bytes
        LDY     #$00
        JSR     CIOPUT
        JSR     PRINT_ERROR

DO_SAVE_QUIT:
        ;LDX     #$10
        LDX     TARGET_IOCB
        JSR     CIOCLOSE

        RTS

        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_SAVE:

;--------------------------------------
OVL_XEP:
;--------------------------------------
        LDY     #$19        ; CMD = $19 (enter 40 col)
        LDX     CMDSEP
        LDA     LNBUF,X
        CMP     #'4'
        BEQ     @+
        DEY                 ; CMD = $18 (enter 80 col)
@:
        LDX     #$00
        TYA
        STA     ICCOM,X
        LDA     #<(OVLBUF-OVL_XEP+EDEV)
        STA     ICBAL,X
        LDA     #>(OVLBUF-OVL_XEP+EDEV)
        STA     ICBAH,X
        LDA     #$2C
        STA     ICAX1,X
        LDA     #$00
        STA     ICAX2,X
        JSR     CIOV
        JMP     DO_CLS

EDEV:   .BYTE   "E:",EOL
        .ALIGN SECTOR_SIZE, $00     ; Align to ATR sector
END_OVL_XEP:

;######################################
;#   MENU MODULE  (loaded on demand)    #
;######################################
;
; The menu is NOT resident.  MENU_LAUNCH reads MENU_CNT sectors from
; MENU_SECT into MENU_RUN (transient RAM above MEMLO, in the unused ATR
; gap after the overlays) and JMPs to MENU_RUN (= MENU_CP).  It is
; assembled at its run address, so it needs no relocation.  A dispatched
; command may overwrite it; the resident MENU_DISP trampoline reloads it.
;######################################
        .ALIGN  SECTOR_SIZE, $00     ; module starts on an ATR sector
MENU_RUN:
;######################################
;#                                    #
;#   DOS 2.0-STYLE MENU FRONT-END     #
;#                                    #
;######################################
;
; Experimental menu interface (branch: menu-experiment).
; Instead of a bare command line, present a lettered menu
; like ATARI DOS 2.0's DUP.SYS.  Each selection assembles a
; normal NOS command line into LNBUF and runs it through the
; existing GETCMDTEST / PARSECMD / DOCMD pipeline, so no
; command logic is duplicated.
;
; Item flag byte:
F_NOARG = $00       ; keyword only, run immediately
F_ARG   = $01       ; keyword + prompt for an argument line
F_DRIVE = $80       ; special: change default drive (Nn:)
F_CLI   = $81       ; special: drop to one classic CLI command

MENU_COUNT = 16     ; items A..P

;--------------------------------------
; Menu command processor (one session).
; Draws the menu, then loops reading item
; selections until warm/cold start leaves.
;--------------------------------------
MENU_CP:
        JSR     MENU_SHOW           ; Clear screen + draw full menu

MENU_SELECT:
        LDA     #<SEL_STR           ; "SELECT ITEM OR RETURN FOR MENU"
        LDY     #>SEL_STR
        LDX     #SEL_LEN
        JSR     MENU_PUTS

        JSR     MENU_READ           ; Read a line; first char -> A
        CMP     #EOL                ; Empty (RETURN) -> redraw menu
        BEQ     MENU_CP

        AND     #$5F                ; Force uppercase
        SEC
        SBC     #'A'                ; Convert letter to 0-based index
        BMI     MENU_SELECT         ; Below 'A' -> ignore
        CMP     #MENU_COUNT
        BCS     MENU_SELECT         ; Past last item -> ignore
        TAX                         ; X = item index
        JSR     MENU_EXEC
        JMP     MENU_SELECT

;--------------------------------------
; Execute the selected menu item (X = idx)
```

```
;---------------------------------------
MENU_EXEC:
        LDA     MENU_FLAGS,X
        CMP     #F_DRIVE
        BNE     @+
        JMP     MENU_DRIVE
@:      CMP     #F_CLI
        BNE     @+
        JMP     MENU_CLI

    ; Normal keyword item.  Save arg flag.
@:      PHA

    ; Copy keyword into LNBUF; length -> Y
        LDA     MENU_KW_L,X
        STA     INBUFF
        LDA     MENU_KW_H,X
        STA     INBUFF+1
        LDY     #$00
MENU_EXEC_CPY:
        LDA     (INBUFF),Y
        CMP     #EOL
        BEQ     MENU_EXEC_KWDONE
        STA     LNBUF,Y
        INY
        BNE     MENU_EXEC_CPY
MENU_EXEC_KWDONE:
        STY     MENU_KLEN           ; keyword length
        PLA                         ; recover arg flag
        BEQ     MENU_EXEC_NOARG

    ;---------------------------------------
    ; Item takes an argument line.
    ; Show a verbose prompt, then read the rest.
    ; (The keyword already sits in LNBUF for the
    ; parser; the prompt is display-only.  X still
    ; holds the item index here.)
    ;---------------------------------------
        LDA     MENU_PR_L,X         ; verbose prompt for this item
        STA     INBUFF
        LDA     MENU_PR_H,X
        STA     INBUFF+1
        JSR     MENU_PUTZ

    ; GETREC the argument into LNBUF + K + 1
    ; (leaving LNBUF[K] as a gap for space or EOL)
        CLC
        LDA     #<LNBUF
        ADC     MENU_KLEN
        STA     ICBAL
        LDA     #>LNBUF
        ADC     #$00
        STA     ICBAH
        INC     ICBAL               ; +1 past the gap
        BNE     @+
        INC     ICBAH
@:      LDA     #$7F
        SEC
        SBC     MENU_KLEN           ; remaining buffer room
        STA     ICBLL
        LDX     #$00                ; CIOV needs X = IOCB #0
        STX     ICBLH
        LDA     #GETREC
        STA     ICCOM
        JSR     CIOV

    ; Empty argument?  (first char at LNBUF[K+1])
        LDY     MENU_KLEN
        INY
        LDA     LNBUF,Y
        CMP     #EOL
        BNE     MENU_EXEC_HASARG

    ; No argument: terminate keyword directly
        LDY     MENU_KLEN
        LDA     #EOL
        STA     LNBUF,Y
        JMP     MENU_DISPATCH

MENU_EXEC_HASARG:
        LDY     MENU_KLEN           ; insert space between keyword and arg
        LDA     #' '
        STA     LNBUF,Y
        JMP     MENU_DISPATCH

MENU_EXEC_NOARG:
        LDY     MENU_KLEN           ; terminate keyword with EOL
        LDA     #EOL
        STA     LNBUF,Y
    ; fall through

;---------------------------------------
; Hand the assembled LNBUF to the resident dispatch trampoline,
; which parses + runs it and then reloads the menu.  (Resident so a
; command that overwrites this transient module can't break us.)
;---------------------------------------
MENU_DISPATCH:
        JMP     MENU_DISP

;---------------------------------------
; Special item: change default drive.
; Assemble LNBUF = 'N' <digit> and let
; DO_DRIVE_CHG validate + apply it.
;---------------------------------------
MENU_DRIVE:
        LDA     #<DRV_STR
        LDY     #>DRV_STR
        LDX     #DRV_LEN
        JSR     MENU_PUTS

        CLC                         ; read digit into LNBUF+1
        LDA     #<LNBUF
        ADC     #$01
        STA     ICBAL
        LDA     #>LNBUF
        ADC     #$00
        STA     ICBAH
        LDA     #$7E
        STA     ICBLL
        LDX     #$00                ; CIOV needs X = IOCB #0
        STX     ICBLH
        LDA     #GETREC
        STA     ICCOM
        JSR     CIOV

        LDA     #'N'                ; LNBUF[0] = 'N'
        STA     LNBUF
        LDA     #CMD_IDX.DRIVE_CHG
        STA     CMD
        JMP     MENU_DISP2          ; resident: DOCMD + reload menu

;---------------------------------------
; Read a line from E: into LNBUF.
; Returns first character in A.
; (The "P. COMMAND LINE" item jumps to the resident MENU_CLI.)
;---------------------------------------
MENU_READ:
        LDX     #$00                ; CIOV needs X = IOCB #0
        STX     ICBLH
        LDA     #<LNBUF
        STA     ICBAL
        LDA     #>LNBUF
        STA     ICBAH
        LDA     #$7F
        STA     ICBLL
        LDA     #GETREC
        STA     ICCOM
        JSR     CIOV
        LDA     LNBUF
        RTS

;---------------------------------------
; PUTCHR X bytes from A/Y to E: (IOCB #0)
; A=addr lo, Y=addr hi, X=length
;---------------------------------------
MENU_PUTS:
        STA     ICBAL
        STY     ICBAH
        STX     ICBLL
        LDX     #$00                ; CIOV needs X = IOCB #0
        STX     ICBLH
        LDA     #PUTCHR
        STA     ICCOM
        JMP     CIOV

;---------------------------------------
; PUTCHR an EOL-terminated string to E:,
; stopping before the EOL (no newline, so
; the reply is typed on the same line).
; INBUFF = string address.
;---------------------------------------
MENU_PUTZ:
        LDY     #$00                ; measure length up to EOL
@:      LDA     (INBUFF),Y
        CMP     #EOL
        BEQ     @+
        INY
        BNE     @-
@:      INY                         ; include the terminating EOL so the
        STY     ICBLL               ; reply starts on a fresh line (length + EOL)
        LDA     INBUFF
        STA     ICBAL
        LDA     INBUFF+1
        STA     ICBAH
        LDX     #$00                ; CIOV needs X = IOCB #0
        STX     ICBLH
        LDA     #PUTCHR
        STA     ICCOM
        JMP     CIOV

;---------------------------------------
; Clear screen and draw the full menu.
; MENU_TXT is a run of EOL-terminated
; lines ending with a $00 sentinel.
;---------------------------------------
MENU_SHOW:
        JSR     DO_CLS
        LDA     #<MENU_TXT
        STA     INBUFF
        LDA     #>MENU_TXT
        STA     INBUFF+1
MENU_SHOW_LOOP:
        LDY     #$00
        LDA     (INBUFF),Y
        BEQ     MENU_SHOW_DONE      ; $00 = end of menu
        LDA     INBUFF
        LDY     INBUFF+1
        JSR     PRINT_STRING        ; prints up to the EOL
        LDY     #$00                ; advance past this line
MENU_SHOW_SCAN:
        LDA     (INBUFF),Y
        INY
        CMP     #EOL
        BNE     MENU_SHOW_SCAN
        TYA
        CLC
        ADC     INBUFF
        STA     INBUFF
        BCC     MENU_SHOW_LOOP
        INC     INBUFF+1
        BNE     MENU_SHOW_LOOP      ; always
MENU_SHOW_DONE:
        RTS

SEL_STR:
        .BYTE   EOL,'SELECT ITEM OR '
        .BYTE   'RETURN'*           ; inverse video, DOS 2.0 style
        .BYTE   ' FOR MENU',EOL
SEL_LEN = *-SEL_STR

DRV_STR:
        .BYTE   EOL,'CHANGE TO DRIVE (1-8)?',EOL
DRV_LEN = *-DRV_STR

MENU_TXT:
        .BYTE   $7D,'FUJINET NETWORK OPERATING SYSTEM 1.0',EOL
        .BYTE   'COPYLEFT 2026 FUJINET',EOL
        .BYTE   EOL
        .BYTE   ' A. DIRECTORY         I. CHANGE DIR',EOL
        .BYTE   ' B. RUN CARTRIDGE     J. SHOW DIR',EOL
        .BYTE   ' C. COPY FILE         K. BINARY SAVE',EOL
        .BYTE   ' D. DELETE FILE(S)    L. BINARY LOAD',EOL
        .BYTE   ' E. RENAME FILE       M. RUN AT ADDR',EOL
        .BYTE   ' F. MAKE DIRECTORY    N. CHANGE DRIVE',EOL
        .BYTE   ' G. REMOVE DIRECTORY  O. TYPE FILE',EOL
        .BYTE   ' H. BASIC ON/OFF      P. COMMAND LINE',EOL
        .BYTE   EOL,EOL,EOL,EOL,EOL
        .BYTE   $00

; Keyword strings assembled for each item (EOL-terminated)
KW_DIR:     .BYTE 'DIR',EOL
KW_CAR:     .BYTE 'CAR',EOL
KW_NCOPY:   .BYTE 'NCOPY',EOL
KW_DEL:     .BYTE 'DEL',EOL
KW_RENAME:  .BYTE 'RENAME',EOL
KW_MKDIR:   .BYTE 'MKDIR',EOL
KW_RMDIR:   .BYTE 'RMDIR',EOL
KW_BASIC:   .BYTE 'BASIC',EOL
KW_NCD:     .BYTE 'NCD',EOL
KW_NPWD:    .BYTE 'NPWD',EOL
KW_SAVE:    .BYTE 'SAVE',EOL
KW_LOAD:    .BYTE 'LOAD',EOL
```

```
KW_RUN:     .BYTE 'RUN',EOL
KW_TYPE:    .BYTE 'TYPE',EOL

; Item tables (index 0..15 == A..P).  KW ptr is
; unused for the two special items (N, P).
MENU_KW_L:
        .BYTE   <KW_DIR             ; A DIRECTORY
        .BYTE   <KW_CAR             ; B RUN CARTRIDGE
        .BYTE   <KW_NCOPY           ; C COPY FILE
        .BYTE   <KW_DEL             ; D DELETE FILE(S)
        .BYTE   <KW_RENAME          ; E RENAME FILE
        .BYTE   <KW_MKDIR           ; F MAKE DIRECTORY
        .BYTE   <KW_RMDIR           ; G REMOVE DIRECTORY
        .BYTE   <KW_BASIC           ; H BASIC ON/OFF
        .BYTE   <KW_NCD             ; I CHANGE DIR
        .BYTE   <KW_NPWD            ; J SHOW DIR
        .BYTE   <KW_SAVE            ; K BINARY SAVE
        .BYTE   <KW_LOAD            ; L BINARY LOAD
        .BYTE   <KW_RUN             ; M RUN AT ADDRESS
        .BYTE   <KW_DIR             ; N CHANGE DRIVE (special)
        .BYTE   <KW_TYPE            ; O TYPE FILE
        .BYTE   <KW_DIR             ; P COMMAND LINE (special)
MENU_KW_H:
        .BYTE   >KW_DIR             ; A
        .BYTE   >KW_CAR             ; B
        .BYTE   >KW_NCOPY           ; C
        .BYTE   >KW_DEL             ; D
        .BYTE   >KW_RENAME          ; E
        .BYTE   >KW_MKDIR           ; F
        .BYTE   >KW_RMDIR           ; G
        .BYTE   >KW_BASIC           ; H
        .BYTE   >KW_NCD             ; I
        .BYTE   >KW_NPWD            ; J
        .BYTE   >KW_SAVE            ; K
        .BYTE   >KW_LOAD            ; L
        .BYTE   >KW_RUN             ; M
        .BYTE   >KW_DIR             ; N
        .BYTE   >KW_TYPE            ; O
        .BYTE   >KW_DIR             ; P
MENU_FLAGS:
        .BYTE   F_ARG               ; A DIRECTORY
        .BYTE   F_NOARG             ; B RUN CARTRIDGE
        .BYTE   F_ARG               ; C COPY FILE
        .BYTE   F_ARG               ; D DELETE FILE(S)
        .BYTE   F_ARG               ; E RENAME FILE
        .BYTE   F_ARG               ; F MAKE DIRECTORY
        .BYTE   F_ARG               ; G REMOVE DIRECTORY
        .BYTE   F_ARG               ; H BASIC ON/OFF
        .BYTE   F_ARG               ; I CHANGE DIR
        .BYTE   F_NOARG             ; J SHOW DIR
        .BYTE   F_ARG               ; K BINARY SAVE
        .BYTE   F_ARG               ; L BINARY LOAD
        .BYTE   F_ARG               ; M RUN AT ADDRESS
        .BYTE   F_DRIVE             ; N CHANGE DRIVE
        .BYTE   F_ARG               ; O TYPE FILE
        .BYTE   F_CLI               ; P COMMAND LINE

; Verbose argument prompts (EOL-terminated, display
; only).  Unused entries (no-arg / special items)
; point at PR_DIR as a harmless placeholder.
PR_DIR:     .BYTE 'DIRECTORY-SEARCH SPEC? ',EOL
PR_NCOPY:   .BYTE 'COPY-FROM,TO? ',EOL
PR_DEL:     .BYTE 'DELETE FILE SPEC? ',EOL
PR_RENAME:  .BYTE 'RENAME-OLD,NEW? ',EOL
PR_MKDIR:   .BYTE 'MAKE DIRECTORY? ',EOL
PR_RMDIR:   .BYTE 'REMOVE DIRECTORY? ',EOL
PR_BASIC:   .BYTE 'BASIC ON OR OFF? ',EOL
PR_NCD:     .BYTE 'CHANGE TO DIRECTORY? ',EOL
PR_SAVE:    .BYTE 'SAVE-NAME,START,END? ',EOL
PR_LOAD:    .BYTE 'BINARY LOAD FILE? ',EOL
PR_RUN:     .BYTE 'RUN AT ADDRESS (HEX)? ',EOL
PR_TYPE:    .BYTE 'TYPE FILE? ',EOL

MENU_PR_L:
        .BYTE   <PR_DIR             ; A DIRECTORY
        .BYTE   <PR_DIR             ; B RUN CARTRIDGE (unused)
        .BYTE   <PR_NCOPY           ; C COPY FILE
        .BYTE   <PR_DEL             ; D DELETE FILE(S)
        .BYTE   <PR_RENAME          ; E RENAME FILE
        .BYTE   <PR_MKDIR           ; F MAKE DIRECTORY
        .BYTE   <PR_RMDIR           ; G REMOVE DIRECTORY
        .BYTE   <PR_BASIC           ; H BASIC ON/OFF
        .BYTE   <PR_NCD             ; I CHANGE DIR
        .BYTE   <PR_DIR             ; J SHOW DIR (unused)
        .BYTE   <PR_SAVE            ; K BINARY SAVE
        .BYTE   <PR_LOAD            ; L BINARY LOAD
        .BYTE   <PR_RUN             ; M RUN AT ADDRESS
        .BYTE   <PR_DIR             ; N CHANGE DRIVE (unused)
        .BYTE   <PR_TYPE            ; O TYPE FILE
        .BYTE   <PR_DIR             ; P COMMAND LINE (unused)
MENU_PR_H:
        .BYTE   >PR_DIR             ; A
        .BYTE   >PR_DIR             ; B
        .BYTE   >PR_NCOPY           ; C
        .BYTE   >PR_DEL             ; D
        .BYTE   >PR_RENAME          ; E
        .BYTE   >PR_MKDIR           ; F
        .BYTE   >PR_RMDIR           ; G
        .BYTE   >PR_BASIC           ; H
        .BYTE   >PR_NCD             ; I
        .BYTE   >PR_DIR             ; J
        .BYTE   >PR_SAVE            ; K
        .BYTE   >PR_LOAD            ; L
        .BYTE   >PR_RUN             ; M
        .BYTE   >PR_DIR             ; N
        .BYTE   >PR_TYPE            ; O
        .BYTE   >PR_DIR             ; P
        .ALIGN  SECTOR_SIZE, $00    ; pad module to whole sectors
MENU_END:
MENU_SECT = MENU_RUN/SECTOR_SIZE - $0D
MENU_CNT  = [MENU_END-MENU_RUN]/SECTOR_SIZE

;#####################################
;#   WILDCARD COPY/DELETE MODULE (load-on-demand) #
;#####################################
; Loaded by WILD_LAUNCH into WILD_RUN.  Assembled AT its run address (no
; relocation), but stored contiguously in the ATR after the menu module;
; the ORG-back restores the file position so the VTOC pad stays correct.
; WILD_MODE (resident, set by the caller) selects the driver: 0 = COPY
; (NCOPY_WILD), 1 = DELETE (NDEL_WILD).  Both share WILD_READDIR/WILD_PREFIX.
WILD_STORE = *                  ; ATR storage address (sector-aligned)
        ORG     WILD_RUN
WILD_ENTRY:
        LDA     WILD_MODE
        BEQ     NCOPY_WILD          ; 0 = copy
        JMP     NDEL_WILD           ; 1 = delete
NCOPY_WILD:
        CLC                         ; INBUFF -> FROM
        LDA     #<LNBUF
        ADC     CMDSEP
        STA     INBUFF
        LDA     #>LNBUF
        ADC     #$00
        STA     INBUFF+1
        JSR     PREPEND_DRIVE       ; INBUFF -> full "Nn:...pattern"
        LDA     INBUFF              ; remember it (PREPEND may have moved it)
        STA     WFROM
        LDA     INBUFF+1
        STA     WFROM+1

        CLC                         ; WTO <- TO (INBUFF -> TO)
        LDA     #<LNBUF
        ADC     CMDSEP+1
        STA     INBUFF
        LDA     #>LNBUF
        ADC     #$00
        STA     INBUFF+1
        JSR     WILD_SAVETO

        LDA     WFROM               ; WPREFIX <- FROM prefix (INBUFF -> FROM)
        STA     INBUFF
        LDA     WFROM+1
        STA     INBUFF+1
        JSR     WILD_PREFIX

        LDA     WFROM               ; read directory (INBUFF -> FROM)
        STA     INBUFF
        LDA     WFROM+1
        STA     INBUFF+1
        JSR     WILD_READDIR
        BCC     NCOPY_WILD_CINIT
        RTS                         ; dir open failed (message shown)

NCOPY_WILD_CINIT:
        LDA     #<WNAMES
        STA     WNPTR
        LDA     #>WNAMES
        STA     WNPTR+1
NCOPY_WILD_CLOOP:
        LDA     WNPTR               ; INBUFF = name cursor
        STA     INBUFF
        LDA     WNPTR+1
        STA     INBUFF+1
        LDY     #$00
        LDA     (INBUFF),Y
        BEQ     NCOPY_WILD_CDONE    ; $00 = end of list
    ; Echo the filename being copied (WNAMES entries are EOL-terminated).
        LDA     WNPTR
        LDY     WNPTR+1
        JSR     PRINT_STRING
        LDA     WNPTR               ; re-point INBUFF (PRINT_STRING may move it)
        STA     INBUFF
        LDA     WNPTR+1
        STA     INBUFF+1
        JSR     WILD_BUILD          ; LNBUF <- NCOPY "<prefix><name>","<TO>"
        LDA     INBUFF              ; save advanced cursor
        STA     WNPTR
        LDA     INBUFF+1
        STA     WNPTR+1
    ; Run it through the normal command pipeline -- the exact path a
    ; user's quoted COPY takes -- so QSTRIP handles spaces and NCOPY's
    ; own Nn:-target filename append does the rest.
        LDA     #$FF
        STA     CMD
        JSR     GETCMDTEST
        JSR     PARSECMD
        JSR     DOCMD
        JMP     NCOPY_WILD_CLOOP
NCOPY_WILD_CDONE:
        RTS

;#####################################
;#   WILDCARD DELETE DRIVER            #
;#####################################
; Mirrors NCOPY_WILD: snapshot every matching name into WNAMES, then walk
; the list.  For each name, prompt "<name> (Y/N)? " and, only on 'Y',
; rebuild a quoted `DEL "<prefix><name>"` and re-run it through the normal
; command pipeline (which lands back in OVL_NDEL -> DO_GENERIC for the
; single delete -- safe, since that reloads OVLBUF, not this $4300 driver).
NDEL_WILD:
    ; Refresh the drive digit in PRMPT (as SHOWPROMPT does).  The menu
    ; front-end never runs SHOWPROMPT, so PRMPT+2 can still be its initial
    ; ' '.  PREPEND_DRIVE injects PRMPT to supply a missing drive, and a
    ; ' ' there yields "N :" -- which GET_DOSDR later folds to unit 0
    ; (space AND $0F = 0), sending the delete to the wrong device.
        LDA     DOSDR
        ORA     #'0'
        STA     PRMPT+2

        CLC                         ; INBUFF -> FROM (arg1)
        LDA     #<LNBUF
        ADC     CMDSEP
        STA     INBUFF
        LDA     #>LNBUF
        ADC     #$00
        STA     INBUFF+1
        JSR     PREPEND_DRIVE       ; INBUFF -> full "Nn:...pattern"
        LDA     INBUFF              ; remember it (PREPEND may have moved it)
        STA     WFROM
        LDA     INBUFF+1
        STA     WFROM+1

        JSR     WILD_PREFIX         ; WPREFIX <- FROM prefix (INBUFF -> FROM)

        LDA     WFROM               ; read directory (INBUFF -> FROM)
        STA     INBUFF
        LDA     WFROM+1
        STA     INBUFF+1
        JSR     WILD_READDIR
        BCC     NDEL_CINIT
        RTS                         ; dir open failed (message shown)

NDEL_CINIT:
        LDA     #<WNAMES
        STA     WNPTR
        LDA     #>WNAMES
        STA     WNPTR+1
NDEL_CLOOP:
        LDA     WNPTR               ; INBUFF = name cursor
        STA     INBUFF
        LDA     WNPTR+1
        STA     INBUFF+1
        LDY     #$00
        LDA     (INBUFF),Y
        BEQ     NDEL_CDONE          ; $00 = end of list

        JSR     NDEL_PROMPT         ; Z=1 if user chose 'Y' (clobbers INBUFF via CIO)
        PHP                         ; save the decision across the rebuild
        LDA     WNPTR               ; re-point INBUFF at the name (CIO moved it)
        STA     INBUFF
        LDA     WNPTR+1
        STA     INBUFF+1
```

```
        JSR     NDEL_BUILD          ; LNBUF <- DEL "<prefix><name>"; advance INBUFF
        LDA     INBUFF              ; save advanced cursor
        STA     WNPTR
        LDA     INBUFF+1
        STA     WNPTR+1
        PLP
        BNE     NDEL_CLOOP          ; not 'Y' -> skip this file

    ; Execute the single delete via the normal pipeline (same path a
    ; user's quoted DEL takes -- QSTRIP handles spaces in the name).
        LDA     #$FF
        STA     CMD
        JSR     GETCMDTEST
        JSR     PARSECMD
        JSR     DOCMD
        JMP     NDEL_CLOOP
NDEL_CDONE:
        RTS

; NDEL_PROMPT: print "<name@INBUFF> (Y/N)? " (no newline) and read a reply
; from E:.  Returns with Z=1 when the reply's first char upper-cases to 'Y'.
; NOTE: CIO (PUTCHR/GETREC) clobbers INBUFF ($F3); the caller must re-point
; it before reusing.  Modeled on MENU_PUTS/MENU_READ.
NDEL_PROMPT:
        LDY     #$00                ; measure name length (up to, excl. EOL)
NDP_LEN:
        LDA     (INBUFF),Y
        CMP     #EOL
        BEQ     NDP_PUTNAME
        INY
        BNE     NDP_LEN
NDP_PUTNAME:
        STY     ICBLL               ; length = name length
        LDA     INBUFF
        STA     ICBAL
        LDA     INBUFF+1
        STA     ICBAH
        LDX     #$00                ; CIOV needs X = IOCB #0
        STX     ICBLH
        LDA     #PUTCHR
        STA     ICCOM
        JSR     CIOV
    ; " (Y/N)? " prompt tail (no EOL, so the reply is typed on the line)
        LDA     #<NDEL_YN
        STA     ICBAL
        LDA     #>NDEL_YN
        STA     ICBAH
        LDA     #NDEL_YN_LEN
        STA     ICBLL
        LDX     #$00
        STX     ICBLH
        LDA     #PUTCHR
        STA     ICCOM
        JSR     CIOV
    ; Read the reply line into TBUF (resident scratch)
        LDA     #<TBUF
        STA     ICBAL
        LDA     #>TBUF
        STA     ICBAH
        LDA     #$08
        STA     ICBLL
        LDX     #$00
        STX     ICBLH
        LDA     #GETREC
        STA     ICCOM
        JSR     CIOV
    ; Decision: upper-case first char, compare 'Y' (empty line -> not 'Y')
        LDA     TBUF
        JSR     TOUPPER
        CMP     #'Y'
        RTS                         ; Z reflects the match

; NDEL_BUILD: LNBUF <- DEL "<WPREFIX><name@INBUFF>" EOL, advancing INBUFF
; past the consumed name (incl. its EOL).  Fully quoted so QSTRIP keeps
; spaces.  Same shape as WILD_BUILD minus the ,"<TO>" middle.
NDEL_BUILD:
        LDX     #$00
        LDY     #$00
NDB_CMD:                            ; 'DEL "'
        LDA     NDEL_CMD,Y
        BEQ     NDB_PFX
        STA     LNBUF,X
        INX
        INY
        BNE     NDB_CMD
NDB_PFX:                            ; <WPREFIX>
        LDY     #$00
NDB_PFX_L:
        LDA     WPREFIX,Y
        CMP     #EOL
        BEQ     NDB_NAME
        STA     LNBUF,X
        INX
        INY
        BNE     NDB_PFX_L
NDB_NAME:                           ; <name>
        LDY     #$00
NDB_NAME_L:
        LDA     (INBUFF),Y
        CMP     #EOL
        BEQ     NDB_NAME_DONE
        STA     LNBUF,X
        INX
        INY
        BNE     NDB_NAME_L
NDB_NAME_DONE:
        INY                         ; consume the EOL; INBUFF += Y
        TYA
        CLC
        ADC     INBUFF
        STA     INBUFF
        BCC     NDB_END
        INC     INBUFF+1
NDB_END:
        LDA     #$22                ; closing quote
        STA     LNBUF,X
        INX
        LDA     #EOL
        STA     LNBUF,X
        RTS

NDEL_CMD:   .BYTE   'DEL ',$22,$00      ; DEL "
NDEL_YN:    .BYTE   ' (Y/N)? '
NDEL_YN_LEN = *-NDEL_YN

; WILD_READDIR: open (INBUFF) as a RAW-format dir; read every matching
; name into WNAMES ($00-terminated).  C clear = ok, C set = open failed.
WILD_READDIR:
        JSR     NEXT_IOCB
        BCC     WRD_OPEN
        RTS                         ; too many files open (message shown)
WRD_OPEN:
        STX     WILD_CH
        LDA     #$03                ; OPEN
        STA     ICCOM,X
        LDA     INBUFF
        STA     ICBAL,X
        LDA     INBUFF+1
        STA     ICBAH,X
        LDA     #$06                ; directory access mode (as DIR uses)
        STA     ICAX1,X
        LDA     #$83                ; DIR_FORMAT::RAW (filename only)
        STA     ICAX2,X
        JSR     CIOV
        BPL     WRD_INIT
        LDA     #<WILD_ERR_STR
        LDY     #>WILD_ERR_STR
        JSR     PRINT_STRING
        SEC
        RTS
WRD_INIT:
    ; Pre-zero the name buffer (512 bytes) so the list is naturally
    ; $00-terminated after the data, then read the WHOLE directory in
    ; one shot: a binary GET BYTES bursts the RAW name stream straight
    ; into WNAMES.  (Names remain EOL-separated; trailing bytes = $00.)
        LDA     #$00
        TAY
WRD_CLR:
        STA     WNAMES,Y
        STA     WNAMES+$100,Y
        INY
        BNE     WRD_CLR
        LDX     WILD_CH
        LDA     #$07                ; GET BYTES
        STA     ICCOM,X
        LDA     #<WNAMES
        STA     ICBAL,X
        LDA     #>WNAMES
        STA     ICBAH,X
        LDA     #$00                ; up to 512 bytes of names
        STA     ICBLL,X
        LDA     #$02
        STA     ICBLH,X
        JSR     CIOV                ; EOF is expected when the dir ends
        LDX     WILD_CH
        JSR     CIOCLOSE
        CLC
        RTS

; WILD_PREFIX: WPREFIX <- (INBUFF) truncated after the last '/' or ':'.
WILD_PREFIX:
        LDY     #$00
WPFX_CPY:
        LDA     (INBUFF),Y
        STA     WPREFIX,Y
        CMP     #EOL
        BEQ     WPFX_SCAN
        INY
        BNE     WPFX_CPY
WPFX_SCAN:
        DEY
        BMI     WPFX_NONE
        LDA     WPREFIX,Y
        CMP     #'/'
        BEQ     WPFX_CUT
        CMP     #':'
        BNE     WPFX_SCAN
WPFX_CUT:
        INY                         ; keep the delimiter
        LDA     #EOL
        STA     WPREFIX,Y
        RTS
WPFX_NONE:
        LDA     #EOL                ; no dir part -> empty prefix
        STA     WPREFIX
        RTS

; WILD_SAVETO: WTO <- (INBUFF).
WILD_SAVETO:
        LDY     #$00
WST_CPY:
        LDA     (INBUFF),Y
        STA     WTO,Y
        CMP     #EOL
        BEQ     WST_DONE
        INY
        BNE     WST_CPY
WST_DONE:
        RTS

; WILD_BUILD: LNBUF <- NCOPY "<WPREFIX><name@(INBUFF)>","<WTO>" EOL,
; advancing INBUFF past the consumed name (incl. its EOL).  Fully quoted
; so GETCMDTEST/QSTRIP preserve spaces in the filename.
WILD_BUILD:
        LDX     #$00
        LDY     #$00
WBLD_CMD:                           ; 'NCOPY "'
        LDA     WILD_CMD,Y
        BEQ     WBLD_PFX
        STA     LNBUF,X
        INX
        INY
        BNE     WBLD_CMD
WBLD_PFX:                           ; <WPREFIX>
        LDY     #$00
WBLD_PFX_L:
        LDA     WPREFIX,Y
        CMP     #EOL
        BEQ     WBLD_NAME
        STA     LNBUF,X
        INX
        INY
        BNE     WBLD_PFX_L
WBLD_NAME:                          ; <name>
        LDY     #$00
WBLD_NAME_L:
        LDA     (INBUFF),Y
        CMP     #EOL
        BEQ     WBLD_NAME_DONE
        STA     LNBUF,X
        INX
        INY
        BNE     WBLD_NAME_L
WBLD_NAME_DONE:
        INY                         ; consume the EOL; INBUFF += Y
        TYA
        CLC
        ADC     INBUFF
        STA     INBUFF
        BCC     WBLD_MID
        INC     INBUFF+1
WBLD_MID:                           ; '","'
```

```
        LDY     #$00
WBLD_MID_L:
        LDA     WILD_MID,Y
        BEQ     WBLD_TO
        STA     LNBUF,X
        INX
        INY
        BNE     WBLD_MID_L
WBLD_TO:                                ; <WTO>
        LDY     #$00
WBLD_TO_L:
        LDA     WTO,Y
        CMP     #EOL
        BEQ     WBLD_END
        STA     LNBUF,X
        INX
        INY
        BNE     WBLD_TO_L
WBLD_END:
        LDA     #$22                    ; closing quote
        STA     LNBUF,X
        INX
        LDA     #EOL
        STA     LNBUF,X
        RTS

WILD_CMD:   .BYTE   'NCOPY ',$22,$00    ; NCOPY "
WILD_MID:   .BYTE   $22,',',$22,$00     ; ","
WILD_ERR_STR:
        .BYTE   'CANT READ DIR',EOL
        .ALIGN  SECTOR_SIZE, $00 ; pad module to whole sectors
WILD_MOD_END:
        ORG     WILD_STORE+[WILD_MOD_END-WILD_RUN]  ; resume ATR file position
WILD_SECT = WILD_STORE/SECTOR_SIZE - $0D
WILD_CNT  = [WILD_MOD_END-WILD_RUN]/SECTOR_SIZE



; ============================================================
; VTOC and Directory
;

; $10 is the added ATR-header
:($B390-*+HDR-$10) DTA $00
VTOCSTA:
    DTA $02,$BD,$02
VTOCEND:

; Fill the remaining bytes of the VTOC sector
    :($80+VTOCSTA-VTOCEND) DTA $00

DIRSTA:
    DTA $60,$C3,$02,$04,$00,C"0*********"
    DTA $60,$C3,$02,$04,$00,C"1 FujiNet  "
    DTA $60,$C3,$02,$04,$00,C"2 Network  "
    DTA $60,$C3,$02,$04,$00,C"3   OS     "
    DTA $60,$C3,$02,$04,$00,C"4          "
    DTA $60,$C3,$02,$04,$00,C"5  v0.7.3  "
    DTA $60,$C3,$02,$04,$00,C"6          "
    DTA $60,$C3,$02,$04,$00,C"7*********"
    DTA $C0
DIREND:

; Fill the remaining sectors of the directory
;    :($400+DIRSTA-DIREND) DTA $00
    :($400+DIRSTA-DIREND) DTA $A0

; Sectors behind directory
;    :($80*352) DTA $00
    :($80*352) DTA $C0

       END
```

`fujinet.online`

FUJINET
NETWORK OPERATING SYSTEM