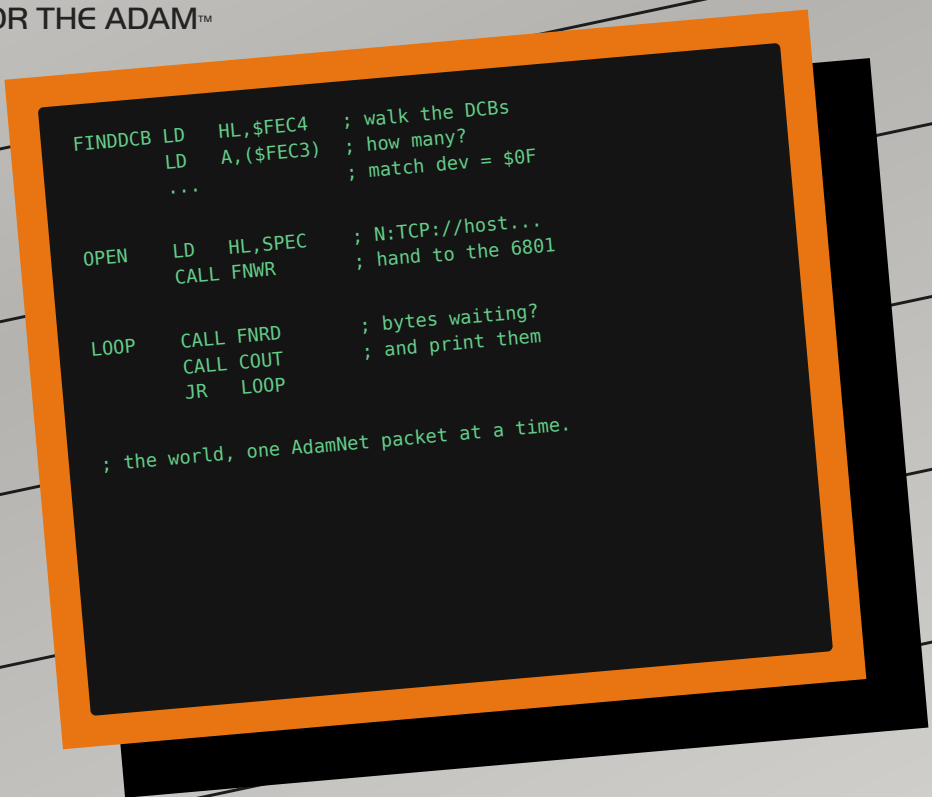
FUJINET

PROGRAMMING GUIDE

PROGRAMMING THE FUJINET

Talking to Your FujiNet from Z80 Assembly Language Over AdamNet — Through EOS, and Directly by DCB Under CP/M

FOR THE ADAM™



A programmer's guide and command reference for the FujiNet WiFi peripheral, from Z80 assembly language.

Free Software

FujiNet's firmware, its client libraries, and this manual are free software, built and given away by a worldwide community of ADAM owners. You may copy this book for a friend — in fact, we would be delighted. Source for everything, this booklet included, lives at `github.com/FujiNetWIFI`.

How This Book Was Verified

Every command code, DCB layout, EOS entry point and error number in this reference was read out of the FujiNet sources, not remembered. The firmware side comes from `fujinet-firmware`: the AdamNet bus in `lib/bus/adamnet/`, the device handlers in `lib/device/adamnet/` (`adamFuji.cpp`, `network.cpp`), and the master command list in `include/fujiCommandID.h`. The ADAM side comes from the EOS C binding (`eos.h`), the `fujinet-lib` ADAM target, and the FujiNet ADAM CP/M library (`fujinet-adam-cpm-lib`), which drives the DCB directly.

Limitation of Warranties

Neither the FujiNet community nor its contributors make any warranty with respect to this manual or to FujiNet. Everything is provided "as is." But unlike 1983, when something bothers you, you can read the source, fix it yourself, and send a pull request.

Trademarks

ADAM, ColecoVision, SmartWRITER, SmartBASIC and EOS are trademarks of their respective owners, used here in tribute. CP/M is a trademark of its owner. Z80 is a trademark of Zilog. FujiNet is a community project, affiliated with none of them.

Conventions

Program listings and DCB dumps are set in a monospaced face. Hexadecimal numbers are written with a leading dollar sign, as the ADAM's own listings write them: `$0F`, `$FEC0`. All assembly is Zilog Z80 mnemonics in the syntax common to `z88dk`'s assembler and `zmac` — labels in column one, a leading `$` on hex — so the examples assemble as-is.

Dedicated to everyone still writing Z80 by hand — and to the memory of the digital data pack, which taught the ADAM to remember, and which FujiNet now teaches to dream.

FUJINET

Programming the FujiNet

A guide to driving the FujiNet WiFi peripheral from Z80 assembly language over AdamNet — through the Elementary Operating System, and by reaching into the Device Control Blocks directly, as a CP/M program must — with a complete reference to the firmware's command set.

This book picks up where *Getting Started with FujiNet CONFIG for the ADAM* leaves off. That book taught your fingers; this one teaches your assembler. By the last chapter you will have written a working *netcat* — open a socket, read it, write it — in a couple of pages of Z80, talking to hardware that did not exist when the Z80 did.

Contents

The Shape of the Thing

A FujiNet is, electrically, a small computer of its own — an ESP32 with WiFi, a memory-card slot, and a wire that pretends to be a peripheral. The pretending is the clever part. To your ADAM, the FujiNet is not a network card and not a co-processor. It is a cluster of **AdamNet devices**: the same intelligent peripheral bus that your keyboard, your disk drives, and your printer already hang from. Everything in this book is, underneath, an AdamNet transaction.

That single fact is what makes the FujiNet so easy to program. You do not install a driver. You do not patch EOS. You ask AdamNet which devices are attached, and you find — alongside the keyboard, the tape drives, and the printer — a handful of devices that were never made of metal:

- **The FujiNet control device** (AdamNet id `$0F`). It mounts disk images, browses hosts, scans WiFi, reads the clock, hashes data, and keeps the slots that CONFIG shows you.
- **The Network devices** (ids `$09` and `$0A`) — the `N:` device. They open TCP, UDP, HTTP, TNFS, FTP, SMB, SSH and TELNET connections and move bytes across them.
- The FujiNet's **disk drives** (ids `$04`–`$07`) and its **printer** (id `$02`) round out the family — ordinary AdamNet block and character devices, mounted from images.

How AdamNet Really Works

AdamNet is a single half-duplex wire running at 62,500 baud. Your Z80 does not drive it directly. A dedicated **master 6801** microcontroller owns the wire; the Z80 and the 6801 meet in shared memory, through a **Peripheral Control Block** (PCB) and, hanging off it, one **Device Control Block** (DCB) per device. You fill in a DCB — here is my buffer, here is its length, now please read (or write) — and poke a byte. The 6801 does the rest: it wraps your bytes into AdamNet packets, clocks them down the wire, collects the reply, and drops a completion code back into the DCB. Chapter 1 derives all of this from the silicon up.

Two Ways In

There are two heights at which you can reach a DCB, and this book teaches both, side by side.

The **EOS road** is the one Coleco intended. The Elementary Operating System — resident in ROM, always at the top of memory — publishes a jump table of device routines. You load a register or two and `CALL` a fixed address; EOS finds

the DCB, fills it, pokes it, and waits. It is the right tool for a SmartBASIC extension, an EOS program, or a cartridge.

The **CP/M road** is the one you must take when EOS is not there. A CP/M program has no jump table to call — but the PCB and its DCBs still sit at `$FEC0`, and the master 6801 is still listening. So you find the DCB yourself, poke it yourself, and spin on the status byte yourself. It costs you a dozen instructions and buys you the FujiNet from inside CP/M, SmartBASIC, or anything else that can address memory.

Every command in this book is shown both ways: the EOS `CALL`, and the direct DCB poke. Learn the two primitives in Chapter 2 and the rest of the book is just payloads.

What You Should Already Know

- Z80 assembly language, and an assembler. The listings use `z88dk` / `zmac` syntax, but they translate to any Z80 assembler without surprises.
- A little of the ADAM's memory map — or a willingness to read Chapter 1, which derives what it needs from scratch.
- Enough of CONFIG (from *Getting Started with FujiNet*) to know what a "host slot" and a "disk slot" are.

How the Reference Is Laid Out

Chapters 1 and 2 build the foundation: how AdamNet is wired, and how a single transaction is shaped — in EOS and in CP/M. Chapters 3 through 5 walk the devices you will actually program — Network, the Fuji control device, and the clock — and every command in them is written up the same way:

EXAMPLE COMMAND

`SEND $F9`

A one-line synopsis, then the payload your program hands the device (the first byte is the command; the rest are its arguments), or the layout of what comes back:

Offset	Bytes	Meaning
0	1	command byte
1	1	an argument byte

RETURNS `$80` from the AdamNet master on success; a device reply you collect with a read. The EOS call and the DCB status code are named here too.

Read the next two chapters in order; after that, dip in wherever you like.

CHAPTER 1

The AdamNet Connection

Before you can send the FujiNet a single command you must understand the three things that stand between your Z80 and the wire: the **master 6801** that owns the bus, the **PCB** that anchors the device list in memory, and the **DCB** through which every transaction passes. This chapter builds them from the bottom. It owes everything to the ADAM's EOS listing and to the FujiNet firmware's `lib/bus/adamnet/`.

What AdamNet Is

AdamNet is a daisy chain of intelligent devices on one three-wire cable: ground, and a single bidirectional data line at 62,500 baud (plus a reset line). Unlike a dumb serial port, every device on it has an **address**, a number from `0` to `15`, and answers only when spoken to. The standard ADAM assigns them like this:

Id	Device
`$01`	keyboard
`$02`	printer
`$04–$07`	disk drives 1–4
`$08`	tape (data pack) drive
`$09–$0A`	FujiNet Network devices (N:)
`$0F`	FujiNet control device

The FujiNet plugs into that chain and adds its own devices to it — the four disk drives it can present, two network devices, a printer, and the control device at `$0F`. To the ADAM they are indistinguishable from real peripherals, which is why no driver is needed.

The Master, the Z80, and Shared Memory

Here is the arrangement that governs everything else. Your Z80 is the ADAM's main processor, but it does **not** bit-bang AdamNet. A second processor — a Motorola 6801, the "master" — sits between the Z80 and the wire and does all the clocking, timing, and error recovery. The two processors rendezvous in the Z80's own RAM:

- A **Peripheral Control Block** (PCB) — a short header the master polls, holding a status byte, a base address, and a count of active devices.
- A **Device Control Block** (DCB) for each device — 21 bytes describing one pending transaction: a status/command byte, a buffer pointer, a length, a block number, the device id, and a few fields the master keeps for itself.

You never touch the wire. You write into a DCB — buffer, length, and a command — and the master, which is forever polling the PCB, notices, carries out the AdamNet transaction, and writes a completion code back into that same status byte. Programming AdamNet is programming the DCB.

The PCB and DCB, Field by Field

Both structures are fixed. This is the EOS binding's view of them, and the CP/M library's, byte for byte:

Off	Bytes	PCB field
0	1	status / command
1	2	base address
3	1	number of active devices (DCBs)

Off	Bytes	DCB field
0	1	status — you poke a request; master writes result
1	2	buffer pointer (your data, low/high)
3	2	length, low/high
5	4	block number (block devices only)
9	1	device number
10	6	reserved for the master
16	1	device id — low nibble is the AdamNet address
18	2	maximum length
20	1	device type / status

IMPORTANT: The byte you care about most is the DCB **status** at offset 0. You write a small number into it to request work (Chapter 2 lists them); the master writes back a code with its high bit set — `$80` and up — when the work is done. That single byte is the whole handshake.

Finding the PCB — and the FujiNet's DCB

Under EOS you rarely hunt for a DCB by hand; the jump table's `FIND_DCB` does it for you (next chapter). But under CP/M there is no EOS, so you find it the hard way — and it is not hard. The PCB lives at a known address, `$FEC0`, and the DCBs follow it immediately, 21 bytes apiece, beginning at `$FEC4`:

Address	Holds
$FEC0	PCB status byte
$FEC3	count of DCBs
$FEC4	first DCB (then every 21 bytes)

To find the FujiNet control device, walk the DCBs and match the low nibble of the device-id byte (offset 16) against `$0F`. This is the CP/M library's `find_dcb`, transliterated to Z80:

Listing 1-1. Find the FujiNet DCB the CP/M way (FINDDCB)

```
DCBBAS  equ  $FEC4          ; first DCB (PCB is $FEC0..$FEC3)
NDCBS   equ  $FEC3          ; count of DCBs, from the PCB
DCBSZ   equ  21             ; bytes per DCB
DEVFUJI equ  $0F            ; FujiNet control device address
;
; FINDDCB - locate the FujiNet control DCB.
;   exit: HL -> DCB, carry CLEAR if found
;         carry SET if no FujiNet on the bus
;
FINDDCB ld   hl,DCBBAS
        ld   a,(NDCBS)
        or   a
        jr   z,FD_NONE      ; no devices at all
        ld   b,a            ; B = how many DCBs to scan
FD_LP   ld   de,16
        add  hl,de          ; HL -> device-id byte (offset 16)
        ld   a,(hl)
        and  $0F            ; low nibble = AdamNet address
        push af
        ld   de,-16
        add  hl,de          ; HL -> back to top of this DCB
        pop  af
        cp   DEVFUJI
        jr   z,FD_OK        ; matched: HL points at its DCB
        ld   de,DCBSZ
        add  hl,de          ; step to the next DCB
        djnz FD_LP
FD_NONE scf                 ; not found
        ret
FD_OK   or   a              ; carry CLEAR = found
        ret
```

NOTE: The network devices are found exactly the same way — match `$09` for `N1:`, `$0A` for `N2:`. The netcat in Appendix C reuses `FINDDCB` with a different address in one register.

With the DCB located, the next chapter shapes an actual transaction — first with EOS, then by poking that DCB directly.

CHAPTER 2

Shaping a Transaction

Every FujiNet command is the same two-beat rhythm: **write** a request to the device (a command byte and its arguments), then **read** the device's reply. Both beats are single AdamNet transactions, and there are two ways to perform each — through EOS, or by poking the DCB. This chapter builds four small routines — `FNWR`, `FNRD` and their CP/M twins — and every later example calls them.

The Write-Then-Read Pattern

A FujiNet command is a character-device **write**: the bytes you send are the command. The first byte is the command code; anything after it is arguments. You do not prefix a length — the length lives in the DCB (or in the register you hand EOS), and the master 6801 adds the AdamNet length header on the wire for you.

To get the answer back, you then **read** the device: the reply lands in your buffer, and the DCB's length field tells you how many bytes arrived. Some commands only write (a mount, a reset); some write then read (a scan, a directory entry, a status). When a reference entry shows a payload **and** a `RETURNS` line, it takes both beats.

Beat	AdamNet	EOS call	DCB status you poke
write	SEND	WR_CH_DEV	3
read	RECEIVE	RD_CH_DEV	4

The EOS Road

EOS keeps a jump table at the very top of memory — a column of fixed addresses you `CALL`. These are the entries this book uses, taken from the EOS binding (`eos.h`) and the EOS listing line numbers beside them:

Addr	EOS routine	In / Out
$FC54	FIND_DCB	A=dev -> IY=DCB, Z=ok
$FC5A	FIND_PCB	-> IY=PCB
$FC7E	REQUEST_DEV_STATUS	A=dev -> IY=DCB
$FCA5	START_RD_CH_DEV	A=dev, DE=buf, BC=len
$FC48	END_RD_CH_DEV	A=dev -> A=status
$FCAE	START_WR_CH_DEV	A=dev, HL=buf, BC=len
$FC51	END_WR_CH_DEV	A=dev -> A=status

A character-device write is a **start** followed by polling **end** until the master reports done. EOS returns the result in `A` a value below `$80` means "still working,"

and $9B means the transaction timed out and should be retried. Listing 2-1 wraps that into FNWR: point FNBUF/FNLEN/FNDEV and call.

Listing 2-1. Write a command through EOS (FNWR)

```
STWRCH  equ  $FCAE          ; START_WR_CH_DEV : A=dev, HL=buf, BC=len
ENWRCH  equ  $FC51          ; END_WR_CH_DEV   : A=dev -> A=status
STRDCH  equ  $FCA5          ; START_RD_CH_DEV : A=dev, DE=buf, BC=len
ENRDCH  equ  $FC48          ; END_RD_CH_DEV   : A=dev -> A=status
;
FNDEV   db   DEVFUJI        ; the device we're talking to
FNBUF   dw   0              ; buffer pointer
FNLEN   dw   0              ; length
;
; FNWR - send (FNLEN) bytes at (FNBUF) to device (FNDEV).
;        exit: A = AdamNet result ($80 = ok).
FNWR    ld   a,(FNDEV)
        ld   hl,(FNBUF)
        ld   bc,(FNLEN)
        call STWRCH         ; start the write (SEND)
FNWRP   ld   a,(FNDEV)
        call ENWRCH         ; poll for completion
        cp   $80
        jr   c,FNWRP        ; < $80 : master still busy
        cp   $9B            ; $9B : general timeout
        jr   z,FNWR         ; re-issue the whole write
        ret                 ; A = $80 (ok) or an error code
```

Reading the reply is the mirror image — START_RD_CH_DEV then poll END_RD_CH_DEV. After it returns, the DCB's length field holds the count of bytes actually delivered into your buffer.

Listing 2-2. Read the reply through EOS (FNRD)

```
; FNRD - read up to (FNLEN) bytes into (FNBUF) from (FNDEV).
;        exit: A = AdamNet result; DCB length = bytes delivered.
FNRD    ld   a,(FNDEV)
        ld   de,(FNBUF)
        ld   bc,(FNLEN)
        call STRDCH         ; start the read (RECEIVE)
FNRDP   ld   a,(FNDEV)
        call ENRDCH         ; poll for completion
        cp   $80
        jr   c,FNRDP        ; still busy
        cp   $9B
        jr   z,FNRD         ; timed out, retry
        ret
```

The CP/M Road

With no EOS jump table, you do exactly what EOS would have done: find the DCB (Listing 1-1), write the buffer pointer and length into it, poke the status

byte with `3` to write or `4` to read, and spin until the master raises the status to `$80` or above. This is the FujiNet CP/M library's `fuji_write` / `fuji_read`, in Z80.

Listing 2-3. Write and read by poking the DCB (DCBWR, DCBRD)

```
; DCBWR - HL -> DCB, DE -> buffer, BC = length. Poke a write.
;         exit: A = result.
DCBWR   push hl
        inc  hl
        ld   (hl),e        ; DCB+1 : buffer low
        inc  hl
        ld   (hl),d        ; DCB+2 : buffer high
        inc  hl
        ld   (hl),c        ; DCB+3 : length low
        inc  hl
        ld   (hl),b        ; DCB+4 : length high
        pop  hl
        ld   (hl),3        ; DCB+0 : status = 3 (write)
DCBWRP  ld   a,(hl)        ; poll the status byte
        cp   $80
        jr   c,DCBWRP      ; < $80 : master still working
        cp   $9B
        jr   z,DCBWR       ; $9B timeout : re-poke (buf/len intact)
        ret
;
; DCBRD - HL -> DCB, DE -> buffer, BC = max length. Poke a read.
DCBRD   push hl
        inc  hl
        ld   (hl),e        ; buffer low
        inc  hl
        ld   (hl),d        ; buffer high
        inc  hl
        ld   (hl),c        ; length low
        inc  hl
        ld   (hl),b        ; length high
        pop  hl
        ld   (hl),4        ; status = 4 (read)
DCBRDP  ld   a,(hl)
        cp   $80
        jr   c,DCBRDP
        cp   $9B
        jr   z,DCBRD
        ret
```

NOTE: The two roads are interchangeable. Every command in the rest of this book is issued with `FNWR`/`FNRD` to run the same example under CP/M, find the DCB once with `FINDDCB` and swap in `DCBWR`/`DCBRD`. The payloads are identical — only the plumbing differs.

Reading the Result

Both roads leave an AdamNet result code in `A`. The master's codes come from the 6801 listing; the ones you will actually meet are below, and Appendix A

lists them all. Anything $80 or higher means the transaction completed; $80 itself is plain success.

Code	Name	Means
$80	ADAMNET_OK	success
$88	SEND_DATA_NACK	device refused the data
$9B	TIMEOUT	no response — retry the transaction

The **device's own** answer is separate: it comes back in the bytes you read. A Network status read, for instance, carries a device error in its fourth byte; a Fuji scan returns a count. Those live with their commands, in the chapters ahead.

Talking to the Right Device

One routine keeps everything pointed at the correct AdamNet address. Set FNDEV before a transaction and both roads obey it. The two you reach for most:

Listing 2-4. Name the devices you'll use

```
DEVFUJI equ  $0F          ; FujiNet control device
DEVNET1 equ  $09          ; N1: network device
DEVNET2 equ  $0A          ; N2: network device
;
USEFUJI ld   a,DEVFUJI
        ld   (FNDEV),a
        ret
USENET1 ld   a,DEVNET1
        ld   (FNDEV),a
        ret
```

With the two primitives in hand, every command in the book reduces to: point FNDEV at the device, lay out the payload, FNWR, and — if there is a reply — FNRD. The next three chapters are just that, command by command.

CHAPTER 3

The Network Device

The Network device — AdamNet id `$09` for `N1:`, `$0A` for `N2:` — is where the FujiNet earns its name. Through it you open TCP and UDP sockets, fetch URLs over HTTP and HTTPS, mount remote filesystems over TNFS, FTP and SMB, and tunnel TELNET and SSH. All of it rides the two primitives from Chapter 2; the protocol is chosen by a **device spec** string, and the verbs are command bytes that happen to be ASCII letters.

The Device Spec

Every network command is addressed to a string of the form

```
N[x]:PROTO://host[:port]/path
```

where `x` is the channel 1–2 on the ADAM, `PROTO` is one of the schemes below, and the rest is the resource. The scheme is matched in the firmware's protocol parser; names are upper-case.

Scheme	Use
TCP	raw TCP socket
UDP	datagram socket
HTTP	web server, cleartext
HTTPS	web server, TLS
TNFS	remote disk filesystem

Scheme	Use
FTP	file transfer
SMB	Windows shares
SSH	secure shell
TELNET	telnet

NOTE: On the ADAM the channel digit chooses the **device**: `N1:` is AdamNet id `$09`, `N2:` is `$0A`. Point `FNDEV` at the matching id before you operate on a channel. There are two network devices, so two connections may be open at once.

Opening and Closing

OPEN

SEND 'O' $4F

Instantiates the protocol named in the device spec and connects. The payload is the command byte, an access **mode**, a translation **mode**, then the spec and a terminating zero.

Offset	Bytes	Value
0	1	$4F ('O')
1	1	access mode (see table)
2	1	translation mode (see table)
3…	N+1	device spec string, NUL-terminated

Mode	Access
$04	read
$08	write
$0C	read/write
$0D	HTTP POST
$05	HTTP DELETE

Trans	Line endings
$00	none (binary)
$01	CR
$02	LF
$03	CR/LF

RETURNS `$80` from the master. Check the device's own error with a `STATUS` afterward. fujinet-lib: `network_open()`.

Listing 3-1. Open a connection (NETOPEN)

```
; NETOPEN - open the spec at (SPEC) on N1: for read/write.
;   the open buffer is [ 'O', mode, trans, spec..., 0 ]
NETOPEN call USENET1        ; FNDEV = $09
        ld   hl,OPBUF
        ld   (FNBUF),hl
        ; build the fixed header
        ld   a,'O'
        ld   (OPBUF),a       ; command
        ld   a,$0C
        ld   (OPBUF+1),a     ; mode = read/write
        xor  a
        ld   (OPBUF+2),a     ; trans = none
        ; copy the NUL-terminated spec after it, counting length
        ld   hl,SPEC
        ld   de,OPBUF+3
        ld   bc,3            ; already 3 header bytes
CPSPEC  ld   a,(hl)
        ld   (de),a
        inc  hl
        inc  de
        inc  bc
        or   a
        jr   nz,CPSPEC       ; stop after copying the NUL
        ld   (FNLEN),bc
        jp   FNWR            ; hand it to the device
;
SPEC    db   "N1:TCP://192.168.1.5:9000/",0
OPBUF   ds   300
```

CLOSE

SEND 'C' $43

Closes the channel, flushes and frees its buffers. A one-byte payload — just the command — is enough; the device knows which channel it is.
RETURNS `$80`. fujinet-lib: `network_close()`.

Listing 3-2. Close a connection (NETCLOSE)

```
NETCLOSE call USENET1
        ld   a,'C'
        ld   (CBUF),a
```

```
        ld   hl,CBUF
        ld   (FNBUF),hl
        ld   hl,1
        ld   (FNLEN),hl
        jp   FNWR
CBUF    db   0
```

Status, Reading, and Writing

These three are the working day of the Network device. `STATUS` tells you how many bytes have arrived and whether the far end is still connected; a read drains them; a write sends.

STATUS

SEND 'S' $53

Write the one-byte command `'S'`, then **read** four bytes back: the pending byte count, a connection flag, and the device error code.

Offset	Bytes	Returned value
0–1	2	bytes waiting to be read (low/high)
2	1	connected: 1 = open, 0 = far end closed
3	1	device error (1 = OK, 136 = EOF)

RETURNS fujinet-lib: `network_status()`.

Listing 3-3. Poll a channel (NETSTAT)

```
; NETSTAT - write 'S', read the 4-byte status into STATBF.
;   after: (BW) = bytes waiting, (CONN), (NERR) set.
NETSTAT call USENET1
        ld   a,'S'
        ld   (SBUF),a
        ld   hl,SBUF
        ld   (FNBUF),hl
        ld   hl,1
        ld   (FNLEN),hl
        call FNWR            ; send the status request
        ld   hl,STATBF
        ld   (FNBUF),hl
        ld   hl,4
        ld   (FNLEN),hl
        call FNRD            ; read the 4-byte reply
        ld   hl,(STATBF)     ; bytes waiting
        ld   (BW),hl
        ld   a,(STATBF+2)
        ld   (CONN),a
        ld   a,(STATBF+3)
        ld   (NERR),a
        ret
SBUF    db   0
STATBF  ds   4
BW      dw   0               ; bytes waiting
```

```
CONN    db   0              ; connection flag
NERR    db   0              ; device error (136 = EOF)
```

READ

RECEIVE

A read takes no command byte — it **is** the RECEIVE beat. Poll `STATUS` first so you ask only for what is waiting; never request more than the device buffer (1024 bytes) holds. The bytes land in your buffer and the DCB length tells you how many came.

RETURNS Data in your buffer; count in the DCB length. fujinet-lib: `network_read()`.

Listing 3-4. Read what's waiting (NETREAD)

```
; NETREAD - read (BW) bytes into RXBUF (caller guarantees BW <= 1024).
NETREAD call USENET1
        ld   hl,RXBUF
        ld   (FNBUF),hl
        ld   hl,(BW)
        ld   (FNLEN),hl
        jp   FNRD           ; data in RXBUF afterward
RXBUF   ds   1024
```

WRITE

SEND 'W' $57

Sends bytes to the channel. The payload is the command `'W'` followed by the data; on the ADAM the firmware pulls the byte count from the AdamNet length the master supplies, so you need only set `FNLEN` to one plus the data length.

RETURNS `$80`. fujinet-lib: `network_write()`.

Listing 3-5. Write a buffer (NETWRITE)

```
; NETWRITE - write B bytes at HL to N1:.  buffer becomes [ 'W',
data... ]
NETWRITE call USENET1
        ld   a,'W'
        ld   (TXBUF),a      ; command byte
        ld   de,TXBUF+1
        ld   c,b            ; save count
        ld   b,0
        push bc
CPTX    ld   a,(hl)         ; copy the data in behind 'W'
        ld   (de),a
        inc  hl
        inc  de
        dec  c
        jr   nz,CPTX
        pop  bc
        inc  bc             ; +1 for the 'W'
```

```
        ld   (FNLEN),bc
        ld   hl,TXBUF
        ld   (FNBUF),hl
        jp   FNWR
TXBUF   ds   256
```

A Complete Exchange: HTTP GET

The routines above are enough to fetch a web page. Open the URL for reading, then poll-and-read until the far end signals EOF — device error `136` in the status reply. This prints the body to the screen through `COUT` (Appendix C shows a console `COUT` for both EOS and CP/M).

Listing 3-6. Fetch a URL and print it

```
GET     call NETOPEN         ; SPEC = "N1:HTTP://...", mode = read
        ret  c               ; open failed
GLOOP   call NETSTAT
        ld   a,(NERR)
        cp   136             ; EOF ?
        jr   z,GDONE
        ld   hl,(BW)
        ld   a,h
        or   l
        jr   z,GLOOP         ; nothing waiting yet, poll again
        ; clamp BW to 1024 if larger (omitted for brevity)
        call NETREAD
        ld   hl,RXBUF
        ld   bc,(BW)
EMIT    ld   a,(hl)
        call COUT            ; print one byte
        inc  hl
        dec  bc
        ld   a,b
        or   c
        jr   nz,EMIT
        jr   GLOOP
GDONE   jp   NETCLOSE
```

WARNING: A read returns at most the device buffer's worth (1024 bytes), so clamp `BW` to 1024 before `NETREAD` when more than that is waiting. The library's `network_read` does this for you; the netcat in Appendix C shows the few extra instructions.

Credentials

For schemes that authenticate — FTP, SMB — set a username and password **before** `OPEN`. Each is a command byte followed by the string.

USERNAME / PASSWORD

SEND $FD / $FE

Payload is the command (`$FD` username, `$FE` password) then the credential string. fujinet-lib: issued as raw writes on the ADAM.

Offset	Bytes	Value
0	1	$FD (user) or $FE (password)
1…	N	credential string

Filesystem Operations

When a channel speaks a filesystem protocol (TNFS, FTP, SMB, even HTTP with WebDAV), a family of command bytes manages files and directories. Each takes the same shape — the command byte, then a device spec naming the target — so one example covers them all.

Code	Char	Operation	fujinet-lib
$21	!	delete file	network_delete
$20		rename (spec is from,to)	network_rename
$23	#	lock (make read-only)	network_lock
$24	$	unlock	network_unlock
$2A	*	make directory	network_mkdir
$2B	+	remove directory	network_rmdir
$2C	,	change directory	network_chdir
$30	0	get current directory	network_getcwd

Listing 3-7. Delete a remote file

```
; delete the file named by (SPEC), e.g. "N1:TNFS://TMA-2/OLD.TXT"
RM      call USENET1
        ld   a,'!'          ; $21 = delete
        ld   (RMBUF),a
        ld   hl,SPEC        ; copy spec in behind the command
        ld   de,RMBUF+1
        ld   bc,1
RMCP    ld   a,(hl)
        ld   (de),a
        inc  hl
        inc  de
        inc  bc
        or   a
        jr   nz,RMCP
        ld   (FNLEN),bc
        ld   hl,RMBUF
        ld   (FNBUF),hl
        jp   FNWR
RMBUF   ds   280
```

Reading JSON

The Network device can parse a JSON document on the FujiNet and hand you single fields, so a 3.58 MHz Z80 never has to. Open the resource, switch the channel into **JSON mode**, parse, then query a path as many times as you like; each query's result is fetched with a `STATUS`+read, exactly like a normal read.

CHANNEL MODE

SEND $FC

Switches the channel between protocol mode (`0`, default) and JSON mode (`1`). Payload: the command byte, then the mode byte.

JSON PARSE

SEND 'P' $50

Parses the document currently waiting on the channel. Command byte only.

JSON QUERY

SEND 'Q' $51

Sets the JSONPath to read; the value is then retrieved with a normal `STATUS`+read.

Offset	Bytes	Value
0	1	$51 ('Q')
1…	N+1	JSONPath, e.g. "/weather/0/main", NUL

RETURNS After the query, `STATUS` reports the value length and a read returns it. fujinet-lib: `network_json_query()`.

HTTP Verbs and Headers

An `HTTP`/`HTTPS` channel is more than a byte pipe. The access mode chosen at `OPEN` selects the verb — read is GET, `$0D` is POST, `$05` is DELETE — and one command byte, `$4D` (`'M'`), steers the channel between the request body and its headers.

HTTP CHANNEL MODE

SEND 'M' $4D

Directs reads and writes on an HTTP channel to the body or the headers.

Value	Meaning
0	body (default)
1	collect request headers
2	read response headers

RETURNS fujinet-lib: `network_http_set_channel_mode()` a header is then sent with an ordinary write.

A raw `TCP:` channel opened read/write is a bidirectional socket — the foundation of the netcat in Appendix C. Two extra command bytes serve the listening and datagram cases.

TCP ACCEPT / CLOSE CLIENT

SEND 'A' $41 / 'c' $63

On a `TCP:` channel opened to listen, `$41` (`'A'`) accepts a waiting client; `$63` (`'c'`) closes the current client while keeping the listener alive. Command byte only.

UDP SET DESTINATION

SEND 'D' $44

For a `UDP:` channel, sets the host:port the next writes are addressed to. Payload: the command byte, then a `"host:port"` string.

RETURNS Companion `$72` (`'r'`, GET REMOTE) reads back the address of the last datagram's sender — handy for replying.

That is the whole Network device. Point `FNDEV` at `$09` or `$0A`, and these bytes give you the Internet. The next chapter turns to the other half of the FujiNet: the control device that manages disks, hosts, and the hardware itself.

CHAPTER 4

The Fuji Control Device

The device at AdamNet id `$0F` is the one CONFIG talks to. It owns the WiFi radio, the list of **hosts** and **disk-image** mounts, the directory browser, persistent app-key storage, the clock, and a drawer of utilities. Where the Network device overloaded ASCII letters as verbs, the Fuji device uses the high-numbered `FUJICMD_` bytes from `fujiCommandID.h`. Set `FNDEV` to `$0F` (call `USEFUJI`); the rhythm from Chapter 2 holds — write the command, then read the reply.

NOTE: Not every code in `fujiCommandID.h` is serviced on the ADAM. This chapter documents those the firmware's `adamFuji.cpp` actually dispatches; Appendix B's table marks every code's status.

The Slots Model

CONFIG presents two arrays you will meet constantly. **Host slots** (8 of them) name the places disks live — a TNFS server, an SMB share, the SD card. **Disk slots** (4 of them) are the drive bays: each remembers a host, an access mode, and a filename, and maps to an AdamNet disk drive `$04`–`$07`. Mounting is the two-step you know from CONFIG: mount a host, browse it, then mount one of its images into a disk slot.

WiFi and the Adapter

SCAN NETWORKS

SEND $FD

Write the one-byte command; the following read returns the count of access points found in the first byte. fujinet-lib: `fuji_scan_for_networks()`.

Listing 4-1. Scan for networks (both roads)

```
; --- the EOS road ---
SCAN    call USEFUJI
        ld   a,$FD
        ld   (SCMD),a
        ld   hl,SCMD
        ld   (FNBUF),hl
        ld   hl,1
        ld   (FNLEN),hl
        call FNWR            ; send SCAN
        ld   hl,RESP
        ld   (FNBUF),hl
        ld   hl,1024
```

```
        ld   (FNLEN),hl
        call FNRD            ; read the reply
        ld   a,(RESP)        ; A = number of APs found
        ret
;
; --- the CP/M road: identical payload, DCB plumbing ---
SCANC   call FINDDCB         ; HL -> Fuji DCB, from Chapter 1
        ret  c
        push hl
        ld   de,SCMD
        ld   bc,1
        call DCBWR           ; poke a write of the 1-byte command
        pop  hl
        ld   de,RESP
        ld   bc,1024
        call DCBRD           ; poke a read of the reply
        ld   a,(RESP)
        ret
SCMD    db   $FD
RESP    ds   1024
```

GET SCAN RESULT

SEND $FC

Write `$FC` followed by the index `n` of the access point you want; the read returns that one's name and signal — a 32-byte SSID and a signed one-byte RSSI. fujinet-lib: `fuji_get_scan_result()`.

SET SSID

SEND $FB

Write `$FB` followed by a 32-byte SSID and a 64-byte password, joining the network. `$FE` (GET SSID) reads the stored pair back. fujinet-lib: `fuji_set_ssid()` / `fuji_get_ssid()`.

GET WIFI STATUS

SEND $FA

Write `$FA` the read returns one byte — `3` = connected, `6` = disconnected. fujinet-lib: `fuji_get_wifi_status()`.

GET ADAPTER CONFIG

SEND $E8

Write `$E8` the read returns the live network configuration in one shot — the joined SSID, hostname, and four-byte IP, gateway, netmask and DNS, plus MAC and BSSID and a firmware-version string.

Offset	Bytes	Field
0	32	SSID
32	64	hostname
96	4	local IP
100	4	gateway
104	4	netmask

108	4	DNS IP
112	6	MAC address
118	6	BSSID
124	15	firmware version string

RETURNS fujinet-lib: `fuji_get_adapter_config()`. The layout is the `AdapterConfig` struct from the CP/M library, byte for byte.

Hosts and Disk Slots

READ / WRITE HOST SLOTS

SEND $F4 / $F3

The eight host slots are an array of eight 32-byte names. Write `$F4` then read all 256 bytes; write `$F3` followed by 256 bytes to store them. fujinet-lib: `fuji_get_host_slots()` / `fuji_put_host_slots()`.

READ / WRITE DEVICE SLOTS

SEND $F2 / $F1

Disk slots are an array of 38-byte records:

Offset	Bytes	Field
0	1	host slot this disk lives on
1	1	access mode (1 = read, 2 = read/write)
2	36	filename

RETURNS Write `$F2` then read the whole array; write `$F1` followed by it to store. The CP/M library models each record as its `DeviceSlot` struct.

MOUNT / UNMOUNT HOST

SEND $F9 / $E6

Brings a host slot online (connects the server) or takes it offline. Payload is the command byte then the host-slot number.

Offset	Bytes	Value
0	1	$F9 (mount) or $E6 (unmount)
1	1	host slot number

RETURNS fujinet-lib: `fuji_mount_host_slot()` / `fuji_unmount_host_slot()`.

MOUNT / UNMOUNT IMAGE

SEND $F8 / $E9

Mounts the disk image recorded in a **disk** slot, making it a live AdamNet drive. Mount takes the slot and an access mode; unmount takes just the slot.

Offset	Bytes	Value
0	1	$F8 (mount) or $E9 (unmount)
1	1	disk slot number
2	1	access mode (mount only: 1=RO, 2=RW)

RETURNS fujinet-lib: `fuji_mount_disk_image()` / `fuji_unmount_disk_image()`. `$D7` (MOUNT ALL, command byte only) mounts every configured slot at once.

Listing 4-2. Mount host 0, then image in disk slot 1

```
MOUNT   call USEFUJI
        ; mount host slot 0
        ld   a,$F9
        ld   (MBUF),a
        xor  a
        ld   (MBUF+1),a      ; host slot 0
        ld   hl,MBUF
        ld   (FNBUF),hl
        ld   hl,2
        ld   (FNLEN),hl
        call FNWR
        ; mount image in disk slot 1, read/write
        ld   a,$F8
        ld   (MBUF),a
        ld   a,1
        ld   (MBUF+1),a      ; disk slot 1
        ld   a,2
        ld   (MBUF+2),a      ; mode = read/write
        ld   hl,3
        ld   (FNLEN),hl
        jp   FNWR
MBUF    ds   4
```

SET DEVICE FILENAME

SEND $E2

Records a filename into a disk slot without mounting: command byte, disk slot, then the filename. To read a slot's filename back, write `$A0 + ds` (so `$A0`…`$A9` for slots 0–9). fujinet-lib: `fuji_set_device_filename()`.

NEW DISK

SEND $E7

Creates a fresh blank image on a host and records it in a disk slot.

Offset	Bytes	Value
0	1	$E7
1	1	host slot
2	1	disk slot
3–6	4	block count (little-endian)
7…	256	filename

RETURNS fujinet-lib: `fuji_create_new()`.

Browsing a Host

To list files on a mounted host, open its directory, read entries one at a time until the end marker, then close.

OPEN DIRECTORY

SEND $F7

Opens a directory on a host slot. Payload: command byte, host slot, then the path and an optional filename filter, separated by a NUL. fujinet-lib: `fuji_open_directory()`.

READ DIR ENTRY

SEND $F6

Write `$F6`, a maximum length to return, and a flags byte; the read returns one entry. Set bit 7 of the flags to append a details block after the name.

Beat	Bytes	Field
write	1	$F6
write	1	max length to return
write	1	flags ($80 = append details)
read	maxlen	filename; details if requested

IMPORTANT: A returned entry whose first byte is `$7F` is the **end-of-directory** marker — stop reading. On the ADAM the firmware also prepends two type-icon bytes to each real filename (a folder, DDP, DSK or ROM glyph); skip or render them as you like.

RETURNS fujinet-lib: `fuji_read_directory()`.

CLOSE DIRECTORY

SEND $F5

Closes the open directory. Command byte only. `$E5` / `$E4` get and set the directory read position for paging. fujinet-lib: `fuji_close_directory()`.

Listing 4-3. List a directory to the screen

```
; host slot 0 already mounted; FNDEV = $0F
LISTDIR call USEFUJI
        ; OPEN DIRECTORY on host 0, path "/"
        ld   a,$F7
        ld   (DBUF),a
        xor  a
        ld   (DBUF+1),a      ; host slot 0
        ld   a,'/'
        ld   (DBUF+2),a      ; path = "/"
        xor  a
        ld   (DBUF+3),a      ; NUL: no filter
        ld   hl,DBUF
        ld   (FNBUF),hl
        ld   hl,4
        ld   (FNLEN),hl
        call FNWR
LD_NEXT ; READ DIR ENTRY, max 40 chars, name only
        ld   a,$F6
        ld   (DBUF),a
        ld   a,40
        ld   (DBUF+1),a      ; max length
```

```
        xor  a
        ld   (DBUF+2),a      ; flags = 0 (no details)
        ld   hl,DBUF
        ld   (FNBUF),hl
        ld   hl,3
        ld   (FNLEN),hl
        call FNWR
        ld   hl,ENTRY        ; read one entry
        ld   (FNBUF),hl
        ld   hl,40
        ld   (FNLEN),hl
        call FNRD
        ld   a,(ENTRY)
        cp   $7F             ; end-of-directory?
        jr   z,LD_END
        ld   hl,ENTRY+2      ; skip the two icon bytes
PR_LP   ld   a,(hl)
        or   a
        jr   z,PR_EOL
        call COUT
        inc  hl
        jr   PR_LP
PR_EOL  ld   a,13
        call COUT            ; carriage return
        jr   LD_NEXT
LD_END  ld   a,$F5           ; CLOSE DIRECTORY
        ld   (DBUF),a
        ld   hl,DBUF
        ld   (FNBUF),hl
        ld   hl,1
        ld   (FNLEN),hl
        jp   FNWR
DBUF    ds   8
ENTRY   ds   64
```

App Keys: Saving State

An **app key** is a small block (up to 64 bytes) the FujiNet stores for your program, indexed by a creator id, an app id and a key id — handy for high scores, settings, or a save game. `OPEN` the key first, declaring read or write, then read or write it.

OPEN APPKEY

SEND $DC

Offset	Bytes	Value
0	1	$DC
1–2	2	creator id (little-endian)
3	1	app id
4	1	key id
5	1	mode: 0 = read, 1 = write

READ / WRITE APPKEY

SEND $DD / $DE

After an `OPEN` in read mode, write `$DD` and read the key's bytes. After an `OPEN` in write mode, write `$DE` followed by the data.

RETURNS fujinet-lib: `fuji_read_appkey()` / `fuji_write_appkey()`.

Boot, Devices, and Housekeeping

A handful of short commands round out the device.

Code	Action	Payload after command byte
$D9	enable/disable CONFIG boot	1 byte: toggle
$D6	set boot mode	1 byte: mode
$D5	enable a device	1 byte: device id
$D4	disable a device	1 byte: device id
$D1	device-enabled status	read returns 1 byte
$D7	mount all slots	none
$D8	copy file between hosts	src, dst, spec
$BB	generate a GUID string	read returns text
$D3	random number	read returns 2 bytes
$FF	reset the FujiNet	none

RETURNS fujinet-lib names follow the obvious pattern — `fuji_set_boot_mode()`, `fuji_enable_device()`, `fuji_generate_guid()`, `fuji_reset()`, and so on.

Hashing, Base64, and QR Codes

The firmware can compute MD5, SHA-1, SHA-256 and SHA-512 digests, encode and decode Base64, and render QR codes — useful for content addressing or showing a link on screen. Each is a small state machine: feed input, compute, ask the length, read the result.

Code	Step
$C8	hash: add input data to the buffer
$C7	hash: compute (algorithm in byte 1); clears buffer
$C6	hash: read length of the digest
$C5	hash: read the digest
$C2	hash: clear the buffer
$D0 $CF $CE $CD	base64 encode: input, compute, length, output
$CC $CB $CA $C9	base64 decode: input, compute, length, output
$BC $BD $BE $BF	QR: input, encode, length, output

That is the Fuji control device. Between it and the Network device you can do anything CONFIG can, and a good deal it cannot. One small thing remains — the clock — and then we build the netcat.

CHAPTER 5

Telling Time

An ADAM has never known what time it is. The FujiNet does — it keeps network time and a configured time zone — and unlike the Atari and Apple, which expose a separate clock device, the ADAM reads the clock straight from the Fuji control device with a single command.

Getting the Time

GET TIME

SEND $D2

Point `FNDEV` at `$0F`, write `$D2`, then read seven bytes: century, year, month, day, hour, minute, second — all binary, no parsing. The century byte is `$13` (19 decimal, added to the two-digit year) so that century plus year gives the full four-digit year.

Offset	Bytes	Field
0	1	century (add to year)
1	1	year (0–99)
2	1	month (1–12)
3	1	day
4	1	hour (24h)
5	1	minute
6	1	second

RETURNS The time is in the FujiNet's configured zone. fujinet-lib: `fuji_get_time()` the layout matches the firmware's `adamnet_get_time()`.

Listing 5-1. Read the time into seven bytes

```
GETTIME call USEFUJI
        ld   a,$D2
        ld   (TCMD),a
        ld   hl,TCMD
        ld   (FNBUF),hl
        ld   hl,1
        ld   (FNLEN),hl
        call FNWR            ; send GET TIME
        ld   hl,NOW
        ld   (FNBUF),hl
        ld   hl,7
        ld   (FNLEN),hl
        call FNRD            ; seven bytes into NOW
        ret
```

```
TCMD    db   $D2
NOW     ds   7              ; cent, year, month, day, hour, min, sec
```

NOTE: The time zone itself is a general FujiNet setting, managed from CONFIG and stored in the adapter configuration. Programs that only need to stamp a file or show a clock read the seven bytes above and are done.

With the clock read, every device the ADAM programmer can reach through the FujiNet has been covered. What remains is to put the Network device through its paces — a real program, start to finish.

Error Codes

Three layers of status meet in FujiNet programming on the ADAM. The **AdamNet master** returns a result code in `A` (or in the DCB status byte) from every transaction. **EOS** returns its own file/device errors from the jump table. And the network **device** reports a finer code inside the bytes you read — most importantly `136`, end-of-file.

AdamNet Master Result Codes

From the 6801 master listing; these are what land in the DCB status byte (high bit set) and in `A` from `FNWR`/`FNRD`:

Code	Name	Meaning
$80	ADAMNET_OK	success
$81	READY_TIMEOUT	READY packet not answered in time
$83	SEND_TIMEOUT	SEND packet not answered in time
$85	SEND_DATA_BREAK	device broke off during a send
$88	SEND_DATA_NACK	device NACKed the sent data
$89	RECEIVE_TIMEOUT	RECEIVE packet not answered in time
$8C	RECEIVE_NACK	device NACKed a receive
$8D	CLR_TIMEOUT	device did not send its data in time
$93	STAT_TIMEOUT	STATUS packet not answered in time
$9B	TIMEOUT	general timeout — retry the transaction

EOS Error Codes

Returned by the file and device calls; strip the high bit (`AND $7F`) before comparing:

Code	Meaning
0	no error
1	DCB not found
2	DCB busy
3	DCB idle
5	no file
9	bad file number
10	end of file
13	storage medium full
$9B	device timeout

Network Device Status Codes

Read from byte 3 of a Network `STATUS` reply (Listing 3-3's `NERR`):

Code	Meaning
1	normal — connected, no error
136	end of file — the resource is fully read

Library Error Codes

When you call `fujinet-lib` instead of issuing raw transactions, it folds these into a small device-agnostic set:

Code	Name	Meaning
$00	FN_ERR_OK	no error
$01	FN_ERR_IO_ERROR	an I/O problem with the device
$02	FN_ERR_BAD_CMD	called with bad arguments
$03	FN_ERR_OFFLINE	the device is offline
$04	FN_ERR_WARNING	non-fatal device warning
$05	FN_ERR_NO_DEVICE	no network device present
$FF	FN_ERR_UNKNOWN	an unmapped device error

APPENDIX B

Command Quick Reference

Everything in this book, condensed. Codes are the **first byte** of the payload you write to the device.

Devices and the EOS Jump Table

Id	AdamNet device
$01	keyboard
$02	printer
$04–$07	disk drives 1–4
$08	tape drive
$09–$0A	Network (N1:, N2:)
$0F	Fuji control

Addr	EOS routine
$FC54	FIND_DCB
$FC5A	FIND_PCB
$FCA5	START_RD_CH_DEV
$FC48	END_RD_CH_DEV
$FCAE	START_WR_CH_DEV
$FC51	END_WR_CH_DEV

DCB Status Requests

Poke	Requests
3	write (character device)
4	read (character device)
>= $80	master's completion code (result)

Network Device (id $09 / $0A)

Code	Char	Operation
$4F	O	open connection (mode, trans, spec)
$43	C	close connection
$53	S	channel status (bw, conn, err)
$57	W	write bytes
–		read waiting bytes (RECEIVE beat)
$FD/$FE		set username / password
$FC		channel mode (0 protocol, 1 JSON)
$50	P	JSON parse
$51	Q	JSON query (then status + read)
$2C	,	change directory
$30	0	get current directory
$20		rename (spec is from,to)
$21	!	delete file
$23	#	lock file
$24	$	unlock file

$2A	*	make directory
$2B	+	remove directory
$4D	M	HTTP channel mode (body/headers)
$41	A	TCP accept connection
$63	c	TCP close client
$44	D	UDP set destination
$72	r	UDP get remote address

Fuji Control Device (id $0F)

The commands the ADAM firmware services. Codes not listed here exist in `fujiCommandID.h` but are not dispatched on the ADAM.

Code	Operation
$FF	reset FujiNet
$FE / $FB	get / set SSID
$FD	scan networks (read returns count)
$FC	get scan result n
$FA	get WiFi status
$F9 / $E6	mount / unmount host slot
$F8 / $E9	mount / unmount disk image
$D7	mount all
$F4 / $F3	read / write host slots
$F2 / $F1	read / write device slots
$E2	set device filename
$A0–$A9	get device filename (slot 0–9)
$E7	new (blank) disk
$F7	open directory
$F6	read directory entry
$F5	close directory
$E5 / $E4	get / set directory position
$E8	get adapter config
$DC	open app key
$DD / $DE	read / write app key
$D9	enable CONFIG boot
$D6	set boot mode
$D5 / $D4	enable / disable device
$D1	device enable status
$D8	copy file
$D3	random number
$D2	get time (7 bytes)
$BB	generate GUID
$C8 $C7 $C6 $C5 $C2	hash: input, compute, len, out, clear
$BC $BD $BE $BF	QR: input, encode, length, output

APPENDIX C

netcat in Z80

Here is the program promised on the title page: a *netcat*, written for **CP/M** on the ADAM. It opens a raw TCP connection through the Network device, then pumps bytes both ways — whatever the far end sends is printed to the screen, and whatever you type is sent to the far end — until the connection drops or you press **ESC**.

It is the whole book in one listing, and it takes the **CP/M road** deliberately: no EOS, just the PCB at `$FEC0`, the network DCB found by hand, and CP/M's own BDOS for the console. `FINDDCB` from Chapter 1 and `DCBWR`/`DCBRD` from Chapter 2 do all the FujiNet work; BDOS function 6 does the keyboard and screen.

What It Needs

Assemble the listing below together with `FINDDCB` (Listing 1-1, with its `match` value changed to `$09` for `N1:`) and `DCBWR` / `DCBRD` (Listing 2-3). Change the address in `HOST` to the server you want to reach, assemble to a `.COM`, and run it from the CP/M prompt.

The Program

Listing C-1. FujiNet netcat — CP/M, direct DCB

```
; ============================================================
;  FUJINET NETCAT for the Coleco ADAM, under CP/M
;  socket <-> screen + keyboard, ESC to quit
;  assemble with FINDDCB (match $09), DCBWR, DCBRD
; ============================================================
BDOS    equ  $0005          ; CP/M entry
CONIO   equ  6              ; BDOS direct console I/O
;
        org  $0100          ; CP/M .COM load address
;
NETCAT  call FINDDCB        ; HL -> N1: DCB (match $09)
        jp   c,NODEV        ; no FujiNet on the bus
        ld   (NETDCB),hl    ; remember it
;
; ---- open TCP, read/write, no translation ----
        ld   hl,(NETDCB)
        ld   de,OPENB       ; [ 'O', $0C, $00, "N1:TCP://...",0 ]
        ld   bc,OPENL
        call DCBWR
;
; ---- the pump: drain the socket, then check the keyboard ----
```

```
PUMP    call STATUS         ; how much is waiting? still up?
        ld   a,(CONNF)
        or   a
        jr   z,CLOSED       ; far end hung up
        ld   a,(NERRF)
        cp   136            ; EOF from the resource
        jr   z,CLOSED
        ld   hl,(BWCNT)
        ld   a,h
        or   l
        jr   z,KEYS         ; nothing waiting -> service keyboard
;
        ld   a,h            ; clamp request to 1024 bytes
        cp   4
        jr   c,RDOK
        ld   hl,1024
RDOK    ld   (RDLEN),hl
        ld   hl,(NETDCB)
        ld   de,RXBUF
        ld   bc,(RDLEN)
        call DCBRD          ; pull the bytes in
        ld   hl,(NETDCB)    ; DCB length = bytes delivered
        ld   de,3
        add  hl,de
        ld   c,(hl)
        inc  hl
        ld   b,(hl)         ; BC = count
        ld   hl,RXBUF
EMIT    ld   a,b
        or   c
        jr   z,KEYS
        ld   e,(hl)         ; print one byte via BDOS
        push hl
        push bc
        ld   c,CONIO
        call BDOS
        pop  bc
        pop  hl
        inc  hl
        dec  bc
        jr   EMIT
;
; ---- one key per pass, so the screen stays responsive -------
KEYS    ld   c,CONIO
        ld   e,$FF          ; $FF = input request
        call BDOS
        or   a
        jr   z,PUMP         ; no key -> keep pumping
        cp   $1B            ; ESC ?
        jr   z,QUIT
        ld   (ONEB),a       ; send this one byte, behind a 'W'
        ld   a,'W'
        ld   (WBUF),a
        ld   hl,(NETDCB)
        ld   de,WBUF
        ld   bc,2           ; 'W' + the key
        call DCBWR
```

```
        jr   PUMP
;
; ---- STATUS: write 'S', read 4 bytes, unpack ----
STATUS  ld   hl,(NETDCB)
        ld   de,SCMD        ; "S"
        ld   bc,1
        call DCBWR
        ld   hl,(NETDCB)
        ld   de,STATB
        ld   bc,4
        call DCBRD
        ld   hl,(STATB)
        ld   (BWCNT),hl     ; bytes waiting
        ld   a,(STATB+2)
        ld   (CONNF),a
        ld   a,(STATB+3)
        ld   (NERRF),a
        ret
;
CLOSED  ld   de,BYEMSG
        call PRSTR
QUIT    ld   hl,(NETDCB)    ; close the channel
        ld   de,CLB         ; "C"
        ld   bc,1
        call DCBWR
        rst  0              ; warm boot back to CP/M
NODEV   ld   de,NOMSG
        call PRSTR
        rst  0
;
; ---- PRSTR: print the $-terminated string at DE ----
PRSTR   ld   c,9            ; BDOS print string
        jp   BDOS
;
; ---- data ------------------------------------------------
OPENB   db   'O',$0C,$00,"N1:TCP://192.168.1.5:9000/",0
OPENL   equ  $-OPENB
SCMD    db   "S"
CLB     db   "C"
WBUF    db   0              ; 'W'
ONEB    db   0              ; the keystroke
STATB   ds   4
BWCNT   dw   0
CONNF   db   0
NERRF   db   0
RDLEN   dw   0
NETDCB  dw   0
RXBUF   ds   1024
BYEMSG  db   13,10,"** CONNECTION CLOSED",13,10,"$"
NOMSG   db   13,10,"** NO FUJINET FOUND",13,10,"$"
```

Trying It

On the other end, anything that speaks TCP will do. The classic test is the Unix `netcat` itself, listening on the port you named:

```
A>NETCAT

HELLO FROM THE ADAM
and hello back from your laptop
the quick brown fox jumped over

** CONNECTION CLOSED
A>
```

Type, and your keystrokes cross the room — or the world — and the reply paints onto a 3.58 MHz machine that predates the network it just joined. That is the whole trick of the FujiNet, and now it is yours to program: find the DCB, write the command, read the reply, and the rest is just Z80.

A FujiNet in every home!