

ADAM™ SmartBASIC™

FUJINET REFERENCE SECTION

(APPENDIX D)

INTRODUCING THE FUJINET COMMANDS

FujiNet is a network adapter that plugs into your ADAM's AdamNet bus and connects it to the Internet or to your own local network. This version of SmartBASIC adds fourteen new statements, all beginning with the letter N, that let your programs talk to it directly -- no PEEKs, POKEs or CALLs required.

The fourteen statements come in two families:

CONNECTION commands -- NOPEN, NCLOSE, NREAD, NWRITE, NSTATUS, NJSONPARSE and NJSONQUERY -- move data over a network connection. Each takes a CHANNEL number (1 to 15) as its first argument, so you can hold several connections open at once. Most programs simply use channel 1.

FILE commands -- NCD, NDIR, NMKDIR, NRMDIR, NDEL, NLOAD and NSAVE -- work on files and directories stored on a network server. They take only a network path and always use channel 1.

Every N command may be typed in immediate mode or used inside a program, just like PRINT. (NLOAD and NSAVE are best used in immediate mode, like LOAD and SAVE.)

NETWORK PATHS

A network path names what you want to talk to. It always begins with N: followed by a protocol, a host, and (sometimes) a port and a path on that host:

```
"N:PROTOCOL://HOST[:PORT]/PATH"
```

The protocols available depend on your FujiNet's firmware; the common ones are HTTP, HTTPS, TCP, UDP, TELNET, TNFS and FTP. For example:

```
"N:HTTP://FUJINET.ONLINE/HELLO.TXT"  
"N:TCP://192.168.1.5:9000/"  
"N:TNFS://TMA-3/GAMES/ADVENTURE.BAS"
```

THINGS TO REMEMBER

1. SmartBASIC strings hold at most 255 characters, so NREAD, NWRITE and NJSONQUERY move at most 255 bytes per call. To move more, loop -- see NREAD and the NFUJI program at the end of this appendix.
2. If no FujiNet is attached to your ADAM, any N command stops with ?ILLEGAL QUANTITY ERROR instead of hanging the computer.
3. Always NCLOSE a channel when you are done with it.

QUICK REFERENCE

CONNECTION COMMANDS

NOPEN	c, path\$, mode, trans	open a connection
NCLOSE	c	close it again
NREAD	c, var\$, length	read up to 255 bytes
NWRITE	c, var\$, length	write up to 255 bytes
NSTATUS	c, bw, conn, err	bytes waiting, connected, error code
NJSONPARSE	c	parse reply as JSON
NJSONQUERY	c, query\$, var\$	fetch one JSON value

FILE COMMANDS (always channel 1)

NCD	path\$	set current net directory
NDIR	path\$	show a directory listing
NMKDIR	path\$	make a directory
NRMDIR	path\$	remove an empty directory
NDEL	path\$	delete a file
NLOAD	path\$	load an ASCII program
NSAVE	path\$	save program as ASCII

ARGUMENTS

c	channel number, 1 to 15
path\$	network path: "N:PROTO://HOST[:PORT]/PATH"
mode	4=read 8=write 12=read/write 13=HTTP POST
trans	line-end translation: 0=none 1=CR 2=LF 3=CR/LF

Command
Statement

NOPEN

NOPEN opens a network connection on a channel. Its syntax is:

```
NOPEN <channel>, <path$>, <mode>, <trans>
```

channel = a number from 1 to 15. Every later command that names the same channel talks to this connection.

path\$ = the network path to open, in quotes or in a string variable.

mode = how you intend to use the connection: 4 to read, 8 to write, 12 to read and write (use this for TCP conversations), 13 to make an HTTP POST.

trans = line-ending translation. Different computers end their text lines differently; the ADAM uses a carriage return (CR). Use 0 for no translation (binary data), 1 to translate network line endings to CR, 2 for LF, 3 for CR/LF.

Opening a channel also returns it to plain (protocol) mode if an earlier NJSONPARSE had switched it to JSON mode.

Test Program:

```
10 NOPEN 1, "N:HTTP://FUJINET.ONLINE/HELLO.TXT", 4, 0
20 NSTATUS 1, BW, CONN, ERR
30 PRINT BW; CONN; ERR
40 NCLOSE 1
```

Sample Run:

```
14 1 1
```

Fourteen bytes are waiting to be read, the connection is open, and the error code of 1 means "no error."

Command
Statement

NCLOSE

NCLOSE closes the connection on a channel and lets the FujiNet reclaim it. Its syntax is:

```
NCLOSE <channel>
```

Close every channel you open, as soon as you are done with it. Closing a channel that is already closed is harmless.

Test Program:

```
10 NOPEN 1, "N:HTTP://FUJINET.ONLINE/HELLO.TXT", 4, 0
20 NCLOSE 1
```

Sample Run:

```
NONE
```

(The program opens a connection and closes it again, printing nothing.)

NREAD

NREAD reads incoming bytes from a channel into a string variable. Its syntax is:

```
NREAD <channel>, <var$>, <length>
```

length = the most bytes you want, 1 to 255. After the read, var\$ holds exactly the bytes that arrived -- which may be fewer than you asked for. Use LEN(var\$) to see how many came.

NREAD WAITS until the device has something to send. So that your program never waits forever, follow the golden rule of network reading: check NSTATUS first, and NREAD only when bytes are waiting. The loop below drains a whole document of any size, 255 bytes at a time; it is the pattern to copy into your own programs.

Test Program:

```
10 NOPEN 1,"N:HTTP://FUJINET.ONLINE/HELLO.TXT",4,0
20 NSTATUS 1,BW,CONN,ERR
30 IF BW=0 AND CONN=0 THEN 90
40 IF BW=0 THEN 20
50 L=BW : IF L>255 THEN L=255
60 NREAD 1,A$,L
70 PRINT A$;
80 GOTO 20
90 NCLOSE 1
```

Sample Run:

```
HELLO, ADAM!
```

Command
Statement

NWRITE

NWRITE sends bytes from a string variable out over a channel. Its syntax is:

```
NWRITE <channel>, <var$>, <length>
```

length = how many characters of var\$ to send, 1 to 255. If length is more than LEN(var\$), only LEN(var\$) characters are sent.

The channel must have been opened with a mode that allows writing: 8, 12 or 13.

Test Program:

```
10 NOPEN 1,"N:TCP://192.168.1.5:9000/",12,0
20 B$="HELLO FUJINET"
30 NWRITE 1,B$,LEN(B$)
40 NCLOSE 1
```

Sample Run:

```
NONE
```

(Nothing prints on the ADAM. The words HELLO FUJINET appear on the machine at address 192.168.1.5 that is listening on port 9000.)

Command
Statement

NSTATUS

NSTATUS asks the FujiNet how a channel is doing and puts the answers into three numeric variables that you name. Its syntax is:

```
NSTATUS <channel>, <bw>, <conn>, <err>
```

bw = bytes waiting -- how many bytes you could NREAD right now. When more than 32767 bytes are waiting this number may show as negative; simply treat ANY value that is not 0 as "there is data to read."

conn = 1 while the connection is open, 0 once the other side has finished and disconnected.

err = the FujiNet's error code for the channel. 1 means no error, 136 means end of file; other values are protocol error codes.

When bw is 0 AND conn is 0, the transfer is complete: nothing is left to read and nothing more is coming. That test appears in every well-mannered receive loop (see NREAD).

Test Program:

```
10 NOPEN 1,"N:HTTP://FUJINET.ONLINE/HELLO.TXT",4,0
20 NSTATUS 1,BW,CONN,ERR
30 PRINT "WAITING:";BW
40 PRINT "CONNECTED:";CONN
50 PRINT "ERROR:";ERR
60 NCLOSE 1
```

Sample Run:

```
WAITING: 14
CONNECTED: 1
ERROR: 1
```

Command
Statement

NJSONPARSE

Much of the information on today's Internet is served in a text format called JSON. NJSONPARSE tells the FujiNet to read the whole reply waiting on a channel and parse it as a JSON document. Its syntax is:

```
NJSONPARSE <channel>
```

Use it right after NOPEN, and follow it with NJSONQUERY to pick out the values you want -- the FujiNet does the hard work of understanding the document so your BASIC program doesn't have to.

NJSONPARSE switches the channel into JSON mode: after it, NREAD returns query results rather than the raw document text. To read a document as plain text again, open it afresh with NOPEN.

Test Program:

```
10 NOPEN 1,"N:HTTPS://FUJINET.ONLINE/FUJINET.JSON",4,0
20 NJSONPARSE 1
30 NJSONQUERY 1,"/0/content",C$
40 PRINT C$
50 NCLOSE 1
```

Sample Run:

(The CONTENT field of the document's first record is printed.)

Command Statement

NJSONQUERY

After NJSONPARSE has parsed a document, NJSONQUERY fetches one value out of it into a string variable. Its syntax is:

```
NJSONQUERY <channel>, <query$>, <var$>
```

query\$ = a path into the document, written as names and element numbers separated by slashes. Element numbers start at 0. Given the document

```
{ "news": [ { "title": "ADAM LIVES" },  
             { "title": "FUJINET SHIPS" } ] }
```

the query `"/news/0/title"` fetches ADAM LIVES and `"/news/1/title"` fetches FUJINET SHIPS.

var\$ = the string variable that receives the value (up to 255 characters of it).

Test Program:

```
10 NOPEN 1,"N:HTTPS://FUJINET.ONLINE/FUJINET.JSON",4,0  
20 NJSONPARSE 1  
30 NJSONQUERY 1,"/0/title",T$  
40 NJSONQUERY 1,"/0/content",C$  
50 PRINT T$ : PRINT C$  
60 NCLOSE 1
```

Sample Run:

(The TITLE and CONTENT fields of the document's first record are printed.)

NCD sets your current network directory, just as walking into a room saves you from giving your full street address every time. Its syntax is:

```
NCD <path$>
```

Later network paths that do not name a protocol and host are taken relative to this prefix.

Test Program:

```
10 NCD "N:TNFS://TMA-3/GAMES"
```

Sample Run:

```
NONE
```

(TMA-3 here and on the following pages is the name of a TNFS file server on the local network. After this command, network files can be named relative to the GAMES directory on that server.)

Command
Statement

NDIR

NDIR prints a directory listing from a network file server on your screen -- it is CATALOG for the network. Its syntax is:

```
NDIR <path$>
```

Test Program:

```
10 NDIR "N:TNFS://TMA-3"
```

Sample Run:

```
GAMES/  
UTILS/  
README.TXT  
NFUJI.BAS
```

Command
Statement

NMKDIR

NMKDIR makes a new directory on a network file server.
Its syntax is:

```
NMKDIR <path$>
```

The server must allow you to write there; a protocol like HTTP that has no directories will refuse.

Test Program:

```
10 NMKDIR "N:TNFS://TMA-3/NEWDIR"
```

Sample Run:

```
NONE
```

(A directory named NEWDIR now exists on the server.
Prove it with NDIR "N:TNFS:

Command
Statement

NRMDIR

NRMDIR removes a directory from a network file server.
Its syntax is:

```
NRMDIR <path$>
```

The directory must be empty -- delete its files first
with NDEL.

Test Program:

```
10 NMKDIR "N:TNFS://TMA-3/NEWDIR"  
20 NRMDIR "N:TNFS://TMA-3/NEWDIR"
```

Sample Run:

```
NONE
```

(The directory is created and then removed again.)

Command
Statement

NDEL

NDEL deletes a file from a network file server. Its syntax is:

```
NDEL <path$>
```

Be careful: like DELETE on a data pack, NDEL does not ask "are you sure?"

Test Program:

```
10 NDEL "N:TNFS://TMA-3/OLDFILE.TXT"
```

Sample Run:

```
NONE
```

(OLDFILE.TXT is gone from the server.)

Command
Statement

NLOAD

NLOAD fetches a SmartBASIC program stored as a plain text (ASCII) listing on a network server and enters it, line by line, exactly as if you were typing it. Its syntax is:

```
NLOAD <path$>
```

Because the lines are entered like typed lines, they MERGE with whatever program is already in memory. For a clean load, type NEW first.

Use NLOAD in immediate mode, like LOAD -- not from inside a running program, because it rewrites program memory as it goes.

Test Program:

```
NEW
NLOAD "N:TNFS://TMA-3/NFUJI.BAS"
LIST
```

Sample Run:

```
10 REM *****
20 REM * FUJINET DEMO FOR SMARTBASIC 1.X *
...
```

(The NFUJI program has been fetched from the server and is ready to RUN.)

Command
Statement

NSAVE

NSAVE writes the program in memory to a network server as a plain text (ASCII) listing -- the same form LIST shows you. Its syntax is:

```
NSAVE <path$>
```

A program saved with NSAVE is loaded back with NLOAD. Because the file is ordinary text, it can also be read, printed or edited on any modern computer -- NSAVE and NLOAD together are a bridge between your ADAM and the rest of the world.

Use NSAVE in immediate mode, like SAVE.

Test Program:

```
NSAVE "N:TNFS://TMA-3/MYPROG.BAS"
```

Sample Run:

```
NONE
```

(The program in memory is now stored on the server as MYPROG.BAS.)

THE NFUJI DEMONSTRATION PROGRAM

The program NFUJI.BAS, supplied with this version of SmartBASIC, exercises the whole FujiNet command set in four short, independent parts. RUN executes part one; the other parts are started from immediate mode with GOTO 1000, GOTO 2000 and GOTO 3000. Each part is listed and explained below.

PART ONE: FETCH A DOCUMENT FROM THE WEB (RUN)

```
100 REM --- HTTP GET, PRINT TO SCREEN ---
110 NOPE 1,"N:HTTP://WWW.GNU.ORG/LICENSES/GPL-3.0.
    TXT",4,0
120 NSTATUS 1,BW,CONN,ERR
130 IF BW=0 AND CONN=0 THEN 200
140 IF BW=0 THEN 120
150 L=BW : IF L>255 THEN L=255
160 NREAD 1,A$,L
170 PRINT A$;
180 GOTO 120
200 NCLOSE 1
210 PRINT : PRINT "--- DONE ---"
220 END
```

(Line 110 is a single program line; it is shown folded here to fit the page.)

Line 110 opens channel 1 on a document at the GNU project's web site, mode 4 (read only), translation 0.

Lines 120-180 are the receive loop from the NREAD page, the beating heart of every download. Line 120 asks how things stand. Line 130 spots the finish: nothing waiting AND disconnected means the document is done. Line 140 goes back to ask again if the connection is alive but no bytes have arrived yet. Line 150 clamps the chunk to 255, the most a string can carry. Line 160 reads the chunk; line 170 prints it with a trailing semicolon so PRINT adds no extra line endings of its own -- the carriage returns inside the document provide the line breaks. Line 180 loops for the next chunk.

Lines 200-220 close the channel and sign off.

PART TWO: READ A JSON FEED (GOTO 1000)

```
1010 REM JSON EXAMPLE (RUN: GOTO 1000)
1020 NOPE 1,"N:HTTPS://FUJINET.ONLINE/FUJINET.
    JSON",4,0
1030 NJSONPARSE 1
1040 NJSONQUERY 1,"/0/content",C$
1050 PRINT C$
1060 NCLOSE 1
1070 END
```

(Line 1020 is likewise one program line, folded here.)

Line 1020 opens the FujiNet news feed -- note HTTPS: the FujiNet handles the secure connection by itself. Line 1030 has the FujiNet parse the reply as JSON. Line 1040 asks for one value: element 0 of the

document (its first record), then that record's CONTENT field. The text lands in C\$ and line 1050 prints it. No receive loop is needed -- the FujiNet holds the parsed document and serves up just the value asked for.

PART THREE: A NETWORK FILE SERVER (GOTO 2000)

```
2010 REM TNFS FILESYSTEM EXAMPLE
2020 NCD "N:TNFS://TMA-3"
2030 NDIR "N:TNFS://TMA-3"
2040 NMKDIR "N:TNFS://TMA-3/NEWDIR"
2050 NRMDIR "N:TNFS://TMA-3/NEWDIR"
2060 END
```

TMA-3 is the name of a TNFS file server on the local network; substitute your own server's name or address. Line 2020 makes it the current network directory, line 2030 prints its catalog on the screen, and lines 2040-2050 make a directory and remove it again -- proof of a round trip.

PART FOUR: TALK TO ANOTHER COMPUTER (GOTO 3000)

```
3010 REM WRITE/POST EXAMPLE
3020 NOPEN 1,"N:TCP://192.168.1.5:9000/",12,0
3030 B$="HELLO FUJINET"
3040 NWRITE 1,B$,LEN(B$)
3050 NCLOSE 1
3060 END
```

Line 3020 opens a raw TCP conversation with the machine at 192.168.1.5, port 9000 -- mode 12, because a conversation needs both reading and writing. Line 3040 sends the thirteen characters of B\$. On the other machine, listen with a command such as "nc -l 9000" and watch the ADAM say hello.

FOR THE CURIOUS: BEHIND THE SCENES

You can use every command in this appendix without reading this page. For the technically inclined, here is what happens underneath.

Each channel is an EOS character device on the AdamNet bus: channel 1 is device 9, channel 2 is device 10, and so on up to channel 15, device 23. The N commands drive these devices through EOS itself -- the same operating system calls that run your printer and data packs -- so they coexist peacefully with everything else on the bus.

Every command is a small packet whose first byte says what to do (O to open, C to close, S for status, W to write, and so on), written to the device; replies come back as device reads. AdamNet delivers up to 1024 bytes per read, so SmartBASIC keeps an internal buffer and hands your program up to 255 bytes at a time from it. NSTATUS counts those buffered bytes in with the bytes still on the FujiNet, so the receive loop's arithmetic always comes out right.

AdamNet transfers are retried automatically on timeout, and before any command is attempted, SmartBASIC checks that the device actually exists on the bus -- which is how a missing FujiNet earns a polite ?ILLEGAL QUANTITY ERROR instead of a frozen computer.

The FujiNet itself does the heavy lifting: domain names, TCP, the secure side of HTTPS, JSON parsing -- work far beyond a 3.58 MHz Z80 -- all happen on the adapter, and your ADAM simply reaps the results.

* * *

This appendix documents the FujiNet extension to SmartBASIC 1.x and is a community supplement to the ADAM™ SmartBASIC™ Programming Manual, Revised Edition (Coleco Industries, 1984).