Introduction to FujiNet for ATARI® Users

Thomas Cherryhomes

Introduction to FujiNet for ATARI® Users

Thomas Cherryhomes

Publication date 2024.03.07

Dedication

This book is dedicated to the thousands of FujiNet users, now across multiple platforms. You all have given the FujiNet team the drive and determination to make FujiNet a comprehensive

platform to connect all of our retro-computing and retro-gaming systems together to do new and fun things.

ACKNOWLEDGEMENTS

Many thanks to the dozens of contributors to every single aspect of FujiNet, not limited to the hardware, the software, the firmware, or the documentation. But there are people that need to be mentioned:

- Joe Honold
- Thomas Cherryhomes
- Jeff Piepmeier
- Steve Boswell
- Oscar Fowler
- Mark Fisher
- Jan Krupa
- Benjamin Krein
- Andy Diller

TABLE OF CONTENTS

List of Figures

List of Tables

List of Examples

PREFACE

Thank you for your purchase of a #FujiNet Wi-Fi Network adapter. We hope that this device will provide you with hours of useful enjoyment, and open your ATARI® computer to the wider world of Internet connectivity.

This manual is for new users of #FujiNet to get a quick grasp on its comprehensive feature set, and can be thought of as both a Getting Started manual, as well as a reference guide.

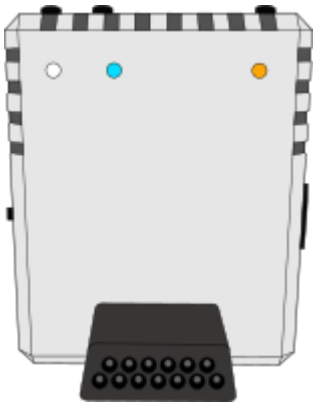
Part I. USER'S GUIDE

TABLE OF CONTENTS

I. What is FujiNet?

FujiNet is a Wi-Fi Network Adapter that plugs into your ATARI® "Peripheral" or SIO port. It provides storage, printing, communications, networking, and more in a single peripheral.

Figure I.1. A FujiNet



Based on the Espressif ESP32® micro controller, FujiNet provides the ATARI® computer with a wide range of devices that may all be used simultaneously. These devices include a virtual "disk" drive, a virtual "printer", a virtual "MODEM", and a "Network Adapter" which provides comprehensive access to a wide variety of local and Internet services through the use of protocol adapters.

Each of these virtual devices is described below.

Virtual Disk Drive "D:"

FujiNet provides the virtual disk device, both so that it can automatically load its configuration program, and to provide the ability to load and use software both from the local SD card slot, or from one of many available TNFS servers that are not only available on the Internet, but can also be run on the local network, as well.

What is TNFS? TNFS is a simple networking protocol that was developed by the Spectranet project, which provides network connectivity to Sinclair® ZX Spectrum™ computers. Because the protocol is open source, and it was deemed useful, the FujiNet project implemented it as the primary method of file sharing for the virtual disk drive.

The FujiNet provides up to eight (8) virtual "disk" drives. These disk drives can accept a disk or file image in one of several available formats (ATR, ATX, or XEX), which can be provided from one of eight (8) host slots.

By selecting one of the host slots in the FujiNet Configuration program (CONFIG), a disk image can be selected, which can then be mounted into one of the eight device slots.

FujiNet can mount one of four different file formats into a device slot:

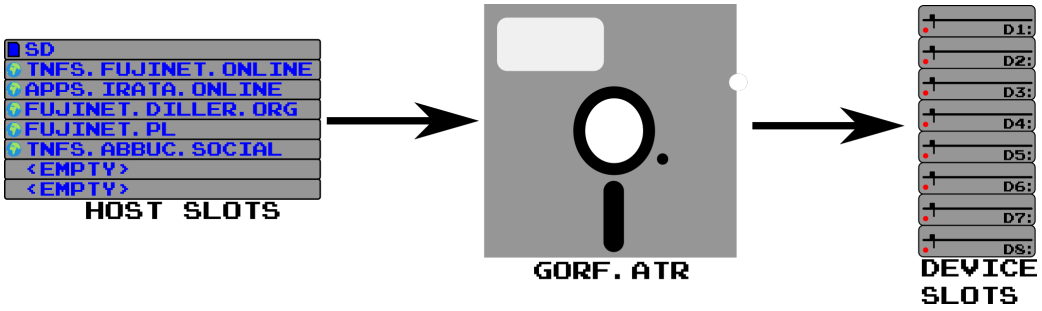
ATR The most common ATARI® disk image format, originally specified by the SIO2PC project. It can support unprotected disks up to 65535 sectors long, and can support disk images with sector sizes of 128 (single density), 256 (double density), and 512 bytes. FujiNet also uses an unused set of three bytes in this header to specify which sectors are allowed to be written

to, when the disk is mounted read only, so that "High Score Enabled" functionality can be supported.

ATX A disk image format specifically designed to represent the unique requirements for copy protected disks on ATARI® systems, by specifying how each track is laid out and positioned on the disk. This includes the alignment of each track. In addition, for each sector, additional information is represented to specify any errors that need to be properly simulated, such as bad, missing, or weak sectors. This format is currently maintained by the a8preservation project: https://www.a8preservation.com/

CAS A cassette image format roughly analogous to ATR. It stores each tape block, along with the delay in milliseconds before the next one should be sent. **CAS files can be mounted into device slot 8.**

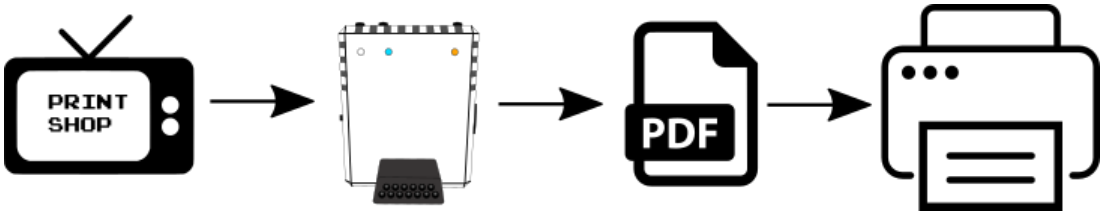
Figure I.2. Virtual Disks



Virtual Printer "P:"

FujiNet also provides a virtual printer device, which works exactly the same as a physical printer attached directly to the ATARI® computer. When a program sends something to the "P:" device, the FujiNet will simulate one of more than a dozen available virtual printers. The end result is rendered into a document which can be read in a viewer, or sent to a modern printer.

Figure I.3. Virtual Printer



The following printers are supported:

RAW	The printer data is sent as-is, with no conversion.
TRIM	Like RAW, but everything after an ATASCII end of line is truncated.
ASCII	Like RAW, but every non-ASCII character is ignored.
ATARI® 820™	A 40 column dot matrix impact printer. Used a 5x7 font with ASCII characters. Printed to 3 7/8 inches (98.425mm) width

	paper. Could also print sideways (vertical). Outputs in PDF. Matches the 400/800 systems.
ATARI® 822™	A 40 column thermal printer. Used a 5x7 font with ASCII characters. Printed to 4 7/16" (112.7125mm) width thermal paper. Could also print dot graphics. Outputs in PDF. Matches the 400/800 systems.
ATARI® 825™	An 80 column dot matrix impact printer. Supported normal, elongated, and condensed fonts in 10 and 16.7 characters per inch widths. Could also support reverse line feeds (which could be used, e.g. to make multi-column text in AtariWriter), and other advanced features. Outputs in PDF. Matches the 400/800 systems.
ATARI® 1020™	A plotter which can draw calligraphic graphics using a pen, and one of four colors (red, green, blue, and black). Unlike most of the other printers in this list, this printer outputs in SVG. Matches the XL systems.
ATARI® 1025™	An 80 column dot matrix impact printer. Uses a 5x7 font with ASCII characters. Printed to either single sheets or fan-fold tractor fed paper anywhere from 4.5 to 8.5 inches (114.3 to 215.9mm) wide. Could accept tractor feed or friction fed paper. Could also accept an automatic document feeder attachment. Outputs in PDF. Matches the XL systems.
ATARI® 1027™	An 80 column letter quality printer. The print font is very close to Prestige Elite. Outputs in PDF. Matches the XL systems.
ATARI® 1029™	An 80 column dot matrix impact printer. Used a 5x7 font, with only ASCII characters. Released in limited quantities in Europe. Outputs in PDF. Matches the XL systems.
ATARI® XMM801™	An 80 column dot matrix impact printer. Epson MX80 and FX80 compatible. Contains many font widths. Can handle NLQ (near-letter-quality) mode. Outputs in PDF. Matches the XE systems.
ATARI® XDM121™	An 80 column letter quality daisy wheel printer. Produced letter quality output. Outputs in PDF. Matches the XE systems.
EPSON® MX80™	An 80 column dot matrix impact printer with graphics printing capability, as well as configurable font widths. Widely considered the standard for dot matrix printers in the 1980s. Outputs in PDF. This is the printer to use with The Print Shop.™
OKI Data® OkiMate 10™	A thermal wax printer capable of printing vivid color graphics. Outputs in PDF.
GRANTIC	An implementation of Claus Buchholz's "PRANTIC" screen capture to printer device. Outputs in PNG.
HTML Printer	Useful for printing out program listings, as it embeds the ATASCII font.

Virtual MODEM "R:"

FujiNet also provides a virtual MODEM device by combining emulation for an ATARI® 850™ interface, and a Hayes® compatible modem with AT instruction set. The resulting virtual device is compatible with existing communications programs that use the "R:" device, and can be used to dial hobbyist Bulletin Board Systems (BBS) hosts over raw or TELNET connections.

The following terminal emulator programs have been tested with FujiNet:

- ICE-T
- BobTerm
- Express!
- AMODEM Plus
- ATARI® Telelink I

The MODEM emulation also supports opening a listening socket that can accept a single incoming connection. This allows existing BBS programs to be used as-is to accept TELNET or raw connections from potential "callers."

The following BBS programs have been tested with FujiNet:

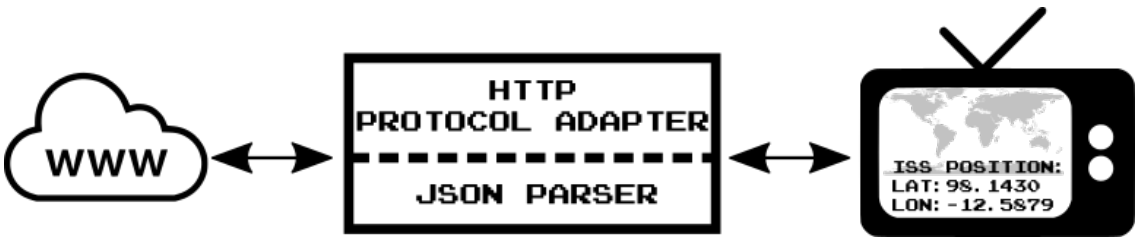
- AMIS BBS
- BBS Express 1.0, 2.0, 5.0
- FoReM-26M, FoReM XE Professional 5.4
- Carina BBS

Network Adapter "N:"

Finally, FujiNet provides a totally new `N:` device for your ATARI® computer. This device brings not only any local network; the wider Internet, accessible from any program or programming language that accepts a devicespec, such as ATARI® BASIC, or AtariWriter™

This is made possible by the addition of protocol adapters which perform all the heavy lifting for the complex protocols used by the modern Internet. These adapters run on the FujiNet itself, and utilize its increased processing power to create an easy to use I/O channel for the ATARI® computer.

Figure I.4. The HTTP Protocol Adapter



Once you have loaded a DOS disk with the "N:" handler, you can, for example, load a BASIC program, directly from a web server:

Example I.1. Loading a BASIC Program from a Web Server

```
READY
RUN"N:HTTP://FUJINET-TESTING.IRATA.ONLINE/BLACKJACK.BAS"
```

Similarly, opening a connection to a TCP socket to host 192.168.0.123 port 1234 is as simple as:

Example I.2. Opening a TCP socket in BASIC

```
READY
OPEN #1,12,2,"N:TCP://192.168.0.123:1234/"
```

- 12 = READ and WRITE
- 2 = Translate Line-feeds into ATASCII EOL (UNIX line endings)

Once a connection is opened, writing to it is equally as simple:

Example I.3. Writing to a TCP socket in BASIC

```
READY
PRINT #1;"HELLO FROM THE ATARI"
```

The EOL here is translated into a UNIX line feed, because of how the connection was previously opened.

Reading from an open connection, is as simple as writing to it:

Example I.4. Reading from a TCP socket in BASIC

```
READY
DIM A$(99)

READY
INPUT #1,A$

READY
PRINT A$
Testing from PC.
```

When done with a connection, it can be closed. FujiNet will then disconnect the socket.

Example I.5. Closing the previously opened TCP socket

```
READY
CLOSE #1
```

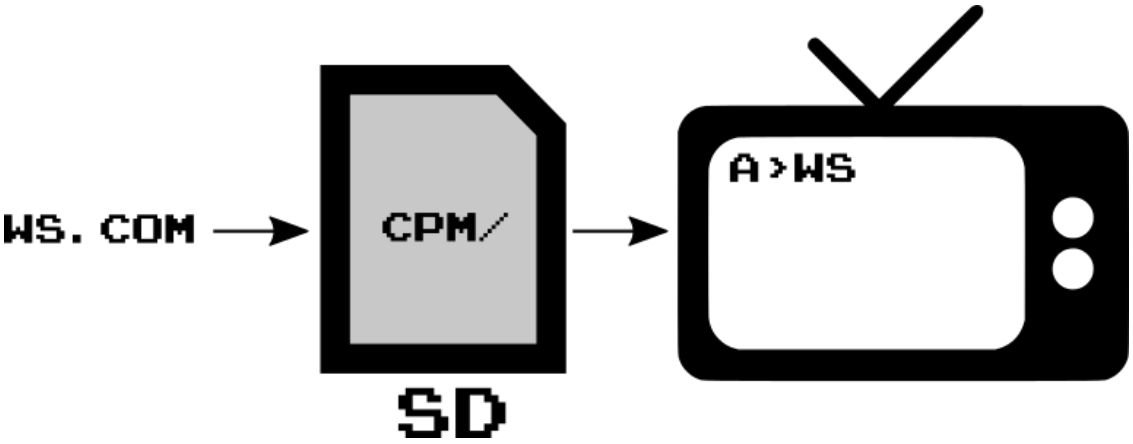
We hope that the addition of the Network device, a whole new class of Internet connected applications can be made, easily and affectively, by anyone in their spare time, using the tools of their choice.

CP/M Compatibility

FujiNet adds a complete Z80 computer emulation that runs a copy of RunCPM. This provides complete CP/M® 2.2 compatibility, being able to run thousands of additional programs, such as WordStar® and dBASE II®.

To make this easier for most users, CP/M files are stored directly on the SD card, without needing to deal with floppy disk image formats. Any CP/M program or data file in question can be copied to your local SD card, placed inside the CPM/ folder, and used.

Figure I.5. CP/M Compatibility



The CP/M emulation uses the ATR8000 protocol for connecting the CP/M computer to the ATARI®. This means that programs such as DT-80 can be used to provide the requisite terminal program. In addition, the CP/M emulation can be accessed by any ATARI® MODEM program that supports the "R:" device.

S.A.M. Voice "P4:"

Because the micro controller provides an 8-bit DAC, and the ATARI® SIO connector provides an AUDIO IN pin, it was trivial to add an implementation of Don't Ask Software's Software Automatic Mouth (S.A.M.) directly on the FujiNet device! This has the advantage of not blanking out the display when you wish to use the voice synthesizer.

Since the P4: device is used, it means that SAM is available to speak at any time, simply by writing to the device.

Example I.6. Making SAM Talk is Easy!

```
READY
OPEN #1,8,0,"P4:"

READY
PRINT #1;"HELLO FROM FUJINET!"
```

APETIME Compatibility

FujiNet includes a real-time clock that is synchronized to the Internet via SNTP (Simple Network Time Protocol), exposed as a device that is compatible with the `APETIME.COM` utility. Operating systems like SpartaDOS and SpartaDOS X can make use of this real time clock to provide time-stamps to files. It is also possible for any program to query the FujiNet to ask for the current time.

II. Unpacking and Connecting

Your Fujinet comes with the following:

- A FujiNet
- A USB Cable
- An SD flash card for local storage.
- This User's Guide
- A Getting Started Guide

III. Configuring your FujiNet

Once the FujiNet has been connected to the ATARI computer, the Wi-Fi Network needs to be configured.

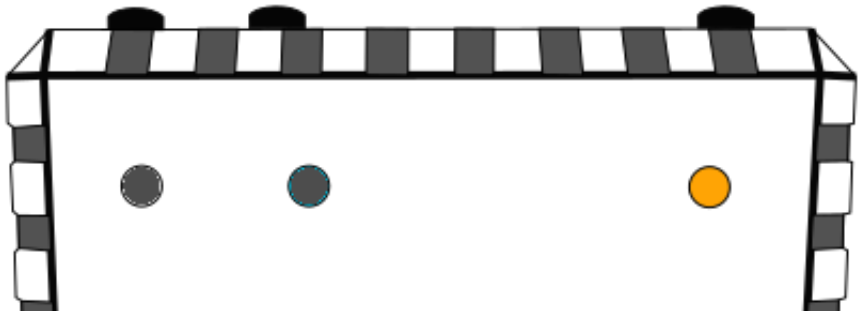
1. Turn on your monitor.
2. Turn on any peripherals you wish to also use.
3. Ensure the power switch on your FujiNet is switched ON.
4. Power on your ATARI® computer.

Note

If a disk drive is configured as drive 1, and is switched on, the FujiNet will defer to the disk drive.

While the FujiNet is responding to the Atari, the yellow activity light will blink.

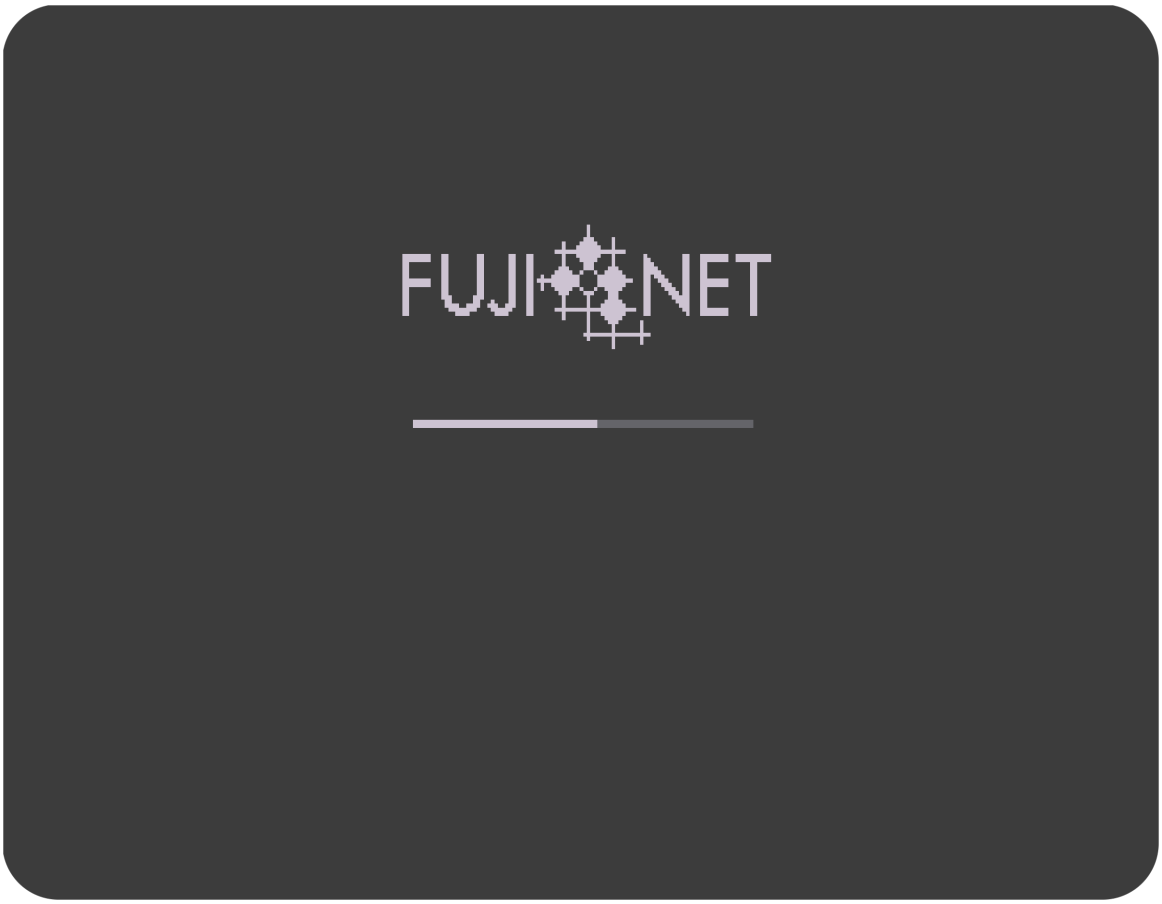
Figure III.1. Top of FujiNet, with Yellow Activity Light



When the FujiNet is first powered on, the CONFIG program will boot. This will allow you to not only set up your wireless network information, but it will also allow you to boot other software from either the SD card, or from the network.

While CONFIG loads, a splash screen is shown with a progress bar:

Figure III.2. FujiNet Loading Screen



Selecting Wi-Fi Network

If FujiNet can not connect to a Wi-Fi Network, it will scan for any nearby networks, and present a selection screen so that the desired network access point, also known as an SSID, may be selected. Each entry will show the name of the network, and the signal strength for each network. More bars indicate a stronger signal. Use the arrow keys, or a joystick connected in port 1 to move the selection bar to the desired network. Press **RETURN** to confirm the selection.

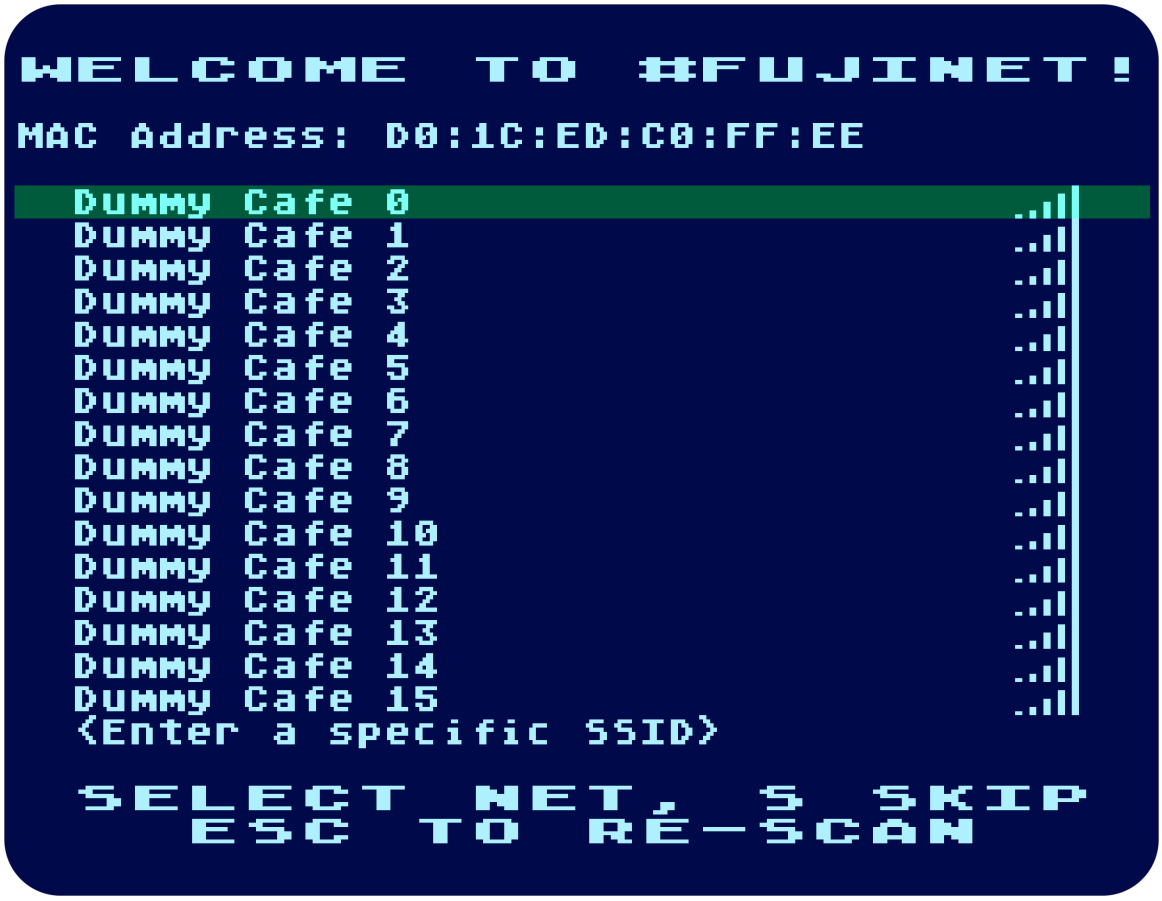
Once a wireless network (SSID) has been confirmed, and successfully connected, CONFIG will proceed to the Host List / Device Slots Screen.

If the network connection is not successful, FujiNet will ask again for a network.

Tip

For best performance, choose the network that is closest to you. This is usually indicated with a high signal strength.

Figure III.3. Selecting Wi-Fi Network



Entering a specific SSID. If the desired network is not broadcasting its SSID, it may be explicitly entered by selecting **<Enter a specific SSID>** and entering the desired SSID, followed by pressing **RETURN**.

Skipping Network Configuration. If the desired network is not available, or only local storage is desired, the network configuration may be skipped by pressing **S**.

Re-Scanning For Networks. Pressing the **ESC** key, will cause FujiNet to re-scan for available networks.

Note

Successful network configurations are saved, and remembered for future use. FujiNet stores the configurations of networks and passwords on both the internal flash memory, and on the SD card.

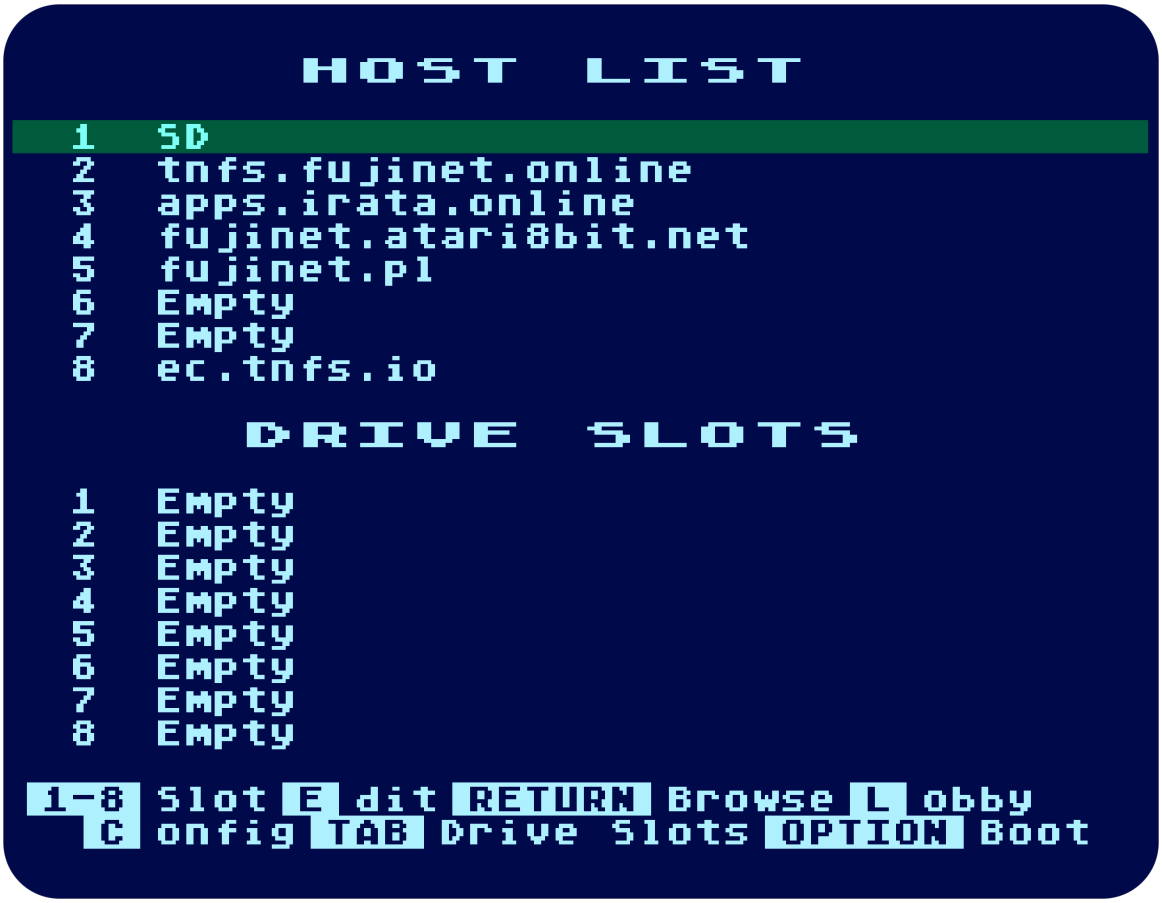
Note

Network configuration can be quickly moved from one FujiNet to another by physically moving the SD card from one unit, to another.

IV. Loading Software

Once the network is successfully configured, or network configuration is skipped, the **HOST LIST / DRIVE SLOTS** screen is shown.

Figure IV.1. HOST LIST / DRIVE SLOTS



is HOST LIST Empty?

If your host slots are empty. They may be re-entered using the **Edit** command using the following procedure:

Procedure IV.1. Re-Populating Host Slots

1. Use arrow keys, or numbers **1**-**8** to select a host slot.
2. Press **E** to edit the host slot.
3. Enter the host slot name.
4. Press **RETURN**
5. Repeat the above steps for as many host slots as desired.

Example Host Slots

1. `SD`
2. `FUJINET.ONLINE`
3. `APPS.IRATA.ONLINE`
4. `FUJINET.DILLER.ORG`
5. `FUJINET.PL`
6. `TNFS.ABBUC.SOCIAL`

HOST LIST

The Host list contains 8 host slots, which specify the source of a disk image. As of current writing, this can be one of two types

SD — Naming a host slot SD indicates that this host slot will point to files on the local SD card.

TNFS — Naming a host slot anything else indicates that this host slot points to a TNFS server with the given host name.

Selecting a Host Slot

Host slots can be selected by one of three methods:

Arrows — The Up and Down arrow keys can move the selection bar by one entry.

Joystick — A joystick plugged into port 1 can move the selection bar by one entry.

1-8 — The number keys **1**-**8** can be used to quickly select an entry.

Opening a Host Slot

Pressing RETURN on a selected host slot, will attempt a connection to the host slot, and if successful, will show the files available on the given host slot, starting with the root directory.

DRIVE SLOTS

V. Using the Virtual Printer

VI. Connecting On-Line via the MODEM

VII. Using the Network Adapter

Part II. PROGRAMMER'S GUIDE

TABLE OF CONTENTS

VIII. Atari BASIC

Using the FujiNet Network Adapter under BASIC is possible by booting either a DOS with the NDEV (N:) handler, or by using the FujiNet NOS. Once loaded, an additional device `N:` will be available to open connections to network resources, and to use them.

Loading BASIC Programs from the Network

Any BASIC program stored on the network (on a web server, for example), can be loaded using the **RUN**, **LOAD**, or **ENTER** commands.

```
READY
RUN"N:HTTP://FUJINET-TESTING.IRATA.ONLINE/BLACKJACK.BAS"
```

Opening a Connection

BASIC's OPEN statement can be used to open a connection to a network endpoint. It has the following format:

```
READY
OPEN #iocb,aux1,aux2,"N:PROTO://HOSTNAME:PORT/PATH/TO/FILE"
```

Where:

iocb	an IO Control Block number from 1 to 7.
aux1	4 = read (HTTP GET), 6 = directory (PROPFIND), 8 = write (or POST), 9 = append, 12 = read/write (HTTP GET), 13 = POST
aux2	Translation mode. 0 = none, 1 = CR<->EOL (old MacOS), 2 = LF<->EOL (UNIX), 3 = CR/LF<->EOL (Windows PC)
PROTO	The Network protocol to use, examples include TCP, UDP, HTTP(S), FTP, TNFS, SMB
HOSTNAME	The host name or IP address to connect to.
PORT	Optional. Specify the numbered port for a service (1-65535)
PATH/TO/FILE	Some protocols specify file systems; so a path can be specified here.

Example VIII.1. OPENing a web server document in BASIC

```
OPEN #1,4,0,"N:HTTPS://WWW.ICANHAZIP.COM"
```

Common OPEN Errors

The following errors can be emitted if there is a problem opening the connection:

ERROR- 129	The IOCB you attempted to open, is already in use. The **CLOSE** statement can close the connection.
ERROR- 200	The connection you've attempted has been explicitly refused by the endpoint.

ERROR- 201 The endpoint you attempted to open is not able to be routed.

ERROR- 202 The socket you've attempted to use didn't finish opening the connection. **This is most common for non-existent services.**

Reading from Connection

Once the connection is opened, both the **INPUT** and **GET** statements can be used to read from a network endpoint. They operate exactly as expected, where INPUT will read one line at a time, broken up by the appropriate end of line character (specified by the **aux2** in the **OPEN** statement), and **GET** will retrieve one byte at a time from the input channel.

Note

A connection must be opened with an aux1 of 4, 6, 12, or 13 in order to read from it.

Example VIII.2. Reading One Line at a time in BASIC.

Reading one line at a time can be accomplished via **INPUT** :

```
READY
DIM A$(99)

READY
INPUT #1,A$

READY
PRINT A$
96.112.44.63
```

Example VIII.3. Reading One byte at a time in BASIC

Reading one byte at a time can be accomplished via **GET** :

```
READY
GET #1,A

READY
PRINT A,CHR$(A)
57        9
```

Important

INPUT and GET will block, meaning that they will wait until something is ready to be received. You can check for bytes waiting ahead of time by using the **STATUS** statement, and subsequently reading PEEK(746) and PEEK(747) for number of bytes waiting.

Writing to Connection

With an open connection, the PRINT and PUT statements can be used to write to an open network connection. They operate exactly as expected, where PRINT will send one line at a time to the network connection, and PUT will send one byte at a time to the network connection.

Note

A connection must be opened with an aux1 of 8, 12, or 13 in order to write to it.

Example VIII.4. Writing One Line at a time, in BASIC.

PRINT can write one line of data at a time to a network endpoint.

```
READY
PRINT #1;"HELLO FROM ATARI"
```

Example VIII.5. Writing One Byte at a time, in BASIC.

PUT can write one line of data at a time to a network endpoint.

```
READY
REM PUT A CR AND A LF
PUT #1,13:PUT #1,10
```

Flushing Connection Output

FujiNet will automatically flush the output to a connection if either of the following two conditions are met:

- If an ATASCII end of line character (155 decimal, $9B hexadecimal) is sent, or...
- If more than 127 non end of line characters are sent.

If you need to send data through the channel, and neither of these two conditions are met, the following **XIO** command can be used to force a flush of the buffer:

Example VIII.6. Forcing a Flush of the Output Channel, in BASIC.

```
XIO 15,#1,12,2,"N:"
```

Note

aux1 and aux2 values should match the values you specified during **OPEN**.

Getting Connection Status

A STATUS statement can be used to determine the current status of an open connection. While STATUS requires that a variable be used, this currently returns the last error experienced on the connection, while the four bytes of DVSTAT indicate the current connection status.

Table VIII.1. PEEKable Status Values

PEEK ADDRESS	DESCRIPTION
PEEK(746)	# of bytes waiting (LO)
PEEK(747)	# of bytes waiting (HI)
PEEK(748)	0=Disconnected, 1=Connected

PEEK ADDRESS	DESCRIPTION
PEEK(749)	Last error returned.

of Bytes Waiting

The number of bytes waiting is expressed as a 16-bit number, with the least significant 8 bits first, followed by the most significant 8 bits. This means that the FujiNet will actively report up to 65535 bytes waiting, even though there may be more in the receive buffer.

The following variable assignment can be used to get the total number of bytes waiting.

Example VIII.7. Getting # of bytes waiting

```
STATUS #1,A:BW=PEEK(747)*256+PEEK(746)
```

Closing a Connection

The CLOSE statement can close a connection that has been opened by OPEN. This has the effect of disconnecting the socket.

```
READY
CLOSE #1
```

Opening a Server (Listening) Connection

In addition to creating client connections, FujiNet can also listen for incoming connections on a given port, and hand them off to the ATARI® for subsequent communication. This is done by opening a connection without a host name, but with a port number. Once opened, the FujiNet will listen for connections on the specified port, and the STATUS statement can be used to check for any incoming connections. Once an incoming connection is found, an XIO command may be sent to accept the connection and subsequently communicate with it.

Example VIII.8. Opening a Listening Connection in BASIC

This example opens a TCP connection on port 6502, using IOCB #1, making the socket read and write with aux1 = 12, and translating to and from UNIX line endings with aux2 = 2.

```
READY
OPEN #1,12,2,"TCP://:6502"
```

Note

As of 2024-03-13, TCP is the only protocol that currently supports creating a listening socket.

Checking for a Client Connection

A STATUS statement can be used to check for an incoming connection. For listening sockets, the only appropriate DVSTAT byte is PEEK(748), which can be repeatedly checked via the following BASIC statement, which creates a loop that only breaks when there is a connection waiting:

```
30 STATUS #1,A:IF NOT PEEK(748) THEN 30
```

Accepting a Client Connection

If PEEK(748) from a previous STATUS statement returns a 1, a connection is waiting to be accepted. Sending the following XIO command will accept the connection, and allow it to be used just like a client connection.

Example VIII.9. Accepting a Client Connection, in BASIC.

```
XIO ASC("A"),#1,12,2,"N:"
```

Once a connection is accepted, any subsequent STATUS statements will return the same information according to the "PEEKable Status Values" table. This status should periodically be checked to ensure that the connection is still connected. If the client disconnects, then it is the responsibility of the server program to close the client connection using the XIO command in the next section.

Closing an Accepted Connection

A server can choose to disconnect an accepted connection. This turns the channel back into a Listening Connection, so that another connection may be subsequently accepted.

Example VIII.10. Closing an Accepted Connection, in BASIC.

```
XIO ASC("c"),#1,12,2,"N:"
```

Closing the Server (Listening) Connection

When a program chooses to shut down, it is a good idea to close the listening socket, so that it may be subsequently used for other tasks. A CLOSE statement is sufficient for this. As a side-effect, any connected client will also be disconnected.

Example VIII.11. Closing the Listening Connection, in BASIC.

```
CLOSE #1
```

Example Programs

This section provides listings of example programs that use the N: device.

NETCAT: A Simple Terminal Emulator

NETCAT is a program which acts like a simple dumb terminal emulator. It will read and display any information that appears on the network channel, while also checking for key presses and sending them out the same network channel. This will continue until either the host disconnects, or a network error occurs. In either case, the network channel is closed, and the program ends.

Example VIII.12. NETCAT Program Listing, in BASIC.

```
10 REM THE SIMPLEST DUMB TERMINAL
20 REM POSSIBLE, WITH THE N: DEVICE
100 OPEN #1,12,2,"N:TCP://BBS.FOZZTEXX.NET/"
101 OPEN #2,4,0,"K:"
```

```
102 TRAP 140
110 IF PEEK(764)=255 THEN 120
111 GET #2,K
112 PUT #1,K
113 XIO 15,#1,12,2,"N:"
120 STATUS #1,A
121 BW=PEEK(747)*256+PEEK(746)
122 IF BW=0 THEN 110
130 FOR M=1 TO BW
131 GET #1,C
132 PUT #16,C
133 NEXT M
134 GOTO 110
140 CLOSE #1
150 PRINT "DISCONNECTED"
160 END
```

Lines 10 and 20 are simple remarks indicating the program and its function.

Line 100 opens a TCP connection to BBS.FOZZTEXX.COM. This can be changed to point to something else, even a different protocol.

Line 101 opens the Keyboard device for subsequent input.

Line 102 sets a TRAP for line 140, which will close the connection, e.g. if the host disconnects, or there is an error.

Line 110 Checks for a pressed key, and jumps to line 120 if no key is pressed.

Line 111 If a key is pressed, read it into variable K.

Line 112 Put the resulting key press into the network channel.

Line 113 Flush the resulting character out to the network channel.

Line 120 Get the status of the network channel.

Line 121 Set **BW** to the number of bytes waiting by combining the bytes in PEEK(746) and PEEK(747).

Line 122 If there are no bytes waiting, jump back to line 110.

Line 130 Start a loop, based on the number of bytes waiting, for each byte:

Line 131 Retrieve the byte from the network channel.

Line 132 Put the byte to the screen (channel #16 becomes channel #0, which is the E: device)

Line 133 Close the loop

Line 134 Return to line 110, so we can check for a key press, again, and repeat the whole process.

Line 140 Any error will trap to this line, which closes the network channel.

Line 150 Print "DISCONNECTED" to the screen.

Line 160 End the program. This closes the Keyboard device, too.

MASTODON: Show the newest Mastodon post

MASTODON is an example program that demonstrates not only the HTTPS protocol adapter, but also the built-in JSON parser. A connection to oldbytes.space, a Mastodon server dedicated to retro computing, is opened. Once opened, the newest public post is retrieved. The resulting post is then parsed and queried for the poster's name, the date of the post, and the post's content. All three of these pieces of information are displayed as they're being queried. Once done, the network connection closes, and the program waits approximately 3 minutes, before looping back to the start of the program. The program repeats until the **BREAK** key is pressed.

Another unique aspect of this program is the simplification of not needing to use **STATUS** to get the # of bytes waiting, relying instead on an end-of-file (EOF) error to stop reading data during the loop at line 90.

Example VIII.13. MASTODON Program Listing, in BASIC.

```
0 DIM A$(256)
1 TRAP 91
2 POKE 756,204
10 OPEN #1,12,0,"N:HTTPS://OLDBYTES.SPACE/api/v1/timelines/public?limit=1"
20 XIO 252,#1,12,1,"N:"
30 XIO ASC("P"),#1,12,0,"N:"
40 XIO ASC("Q"),#1,12,3,"N:/0/account/display_name"
50 INPUT #1,A$:? A$
60 XIO ASC("Q"),#1,12,3,"N:/0/created_at"
70 INPUT #1,A$:? A$
80 XIO ASC("Q"),#1,12,3,"N:/0/content"
90 GET #1,A:? CHR$(A);:GOTO 90
91 CLOSE #1:? :?
100 POKE 18,0:POKE 19,0:POKE 20,0
110 IF PEEK(19)<30 THEN 110
120 GOTO 1
```

Line 0 dimensions a 256 byte string variable A$.

Line 1 sets 91 as an error trap.

Line 2 changes the character set base to the International Character set. This line may be omitted on ATARI® 400/800™ systems, which do not have this character set.

Line 10 opens a connection to the oldbytes.space Mastodon server using the HTTPS protocol. The path points to an API call that fetches posts from the public time line. The "?" Indicates a GET query parameter, and in this case limit=1 ensures that the server only sends a single post.

Line 20 uses **XIO** to send a SET CHANNEL MODE command. This changes the channel to allow the '**P**'arse and '**Q**'uery commands that will be used on the proceeding lines.

Line 30 uses **XIO** to send a PARSE command. This reads the network channel until there is no more data. The channel is subsequently passed through the JSON parser, with the results of the parse sitting in FujiNet memory, to be queried using the '**Q**'uery command.

Line 40 uses **XIO** to send a '**Q**'uery command. This particular JSON Query asks for the display name of a given poster. However, it is inside an array, so we have to ask for the first element of the array as part of the query, hence the "/0/"

Line 50 calls INPUT to read the queried result into variable A$, and subsequently display it on screen with the '**?**' statement.

Line 60 uses **XIO** to send a '**Q**'uery command which asks for the created_at key for the very first returned post.

Line 70 calls **INPUT** to read the queried result into variable A$, and subsequently display it.

Line 80 uses **XIO** to send a final '**Q**'uery command, which asks for the content for the very first returned post.

Line 90 Sets up an infinite loop which **GET**s and displays one byte at a time, because **INPUT** can only fetch single lines up to 127 bytes long. The end of the content will generate an end of file (EOF) error, so the **TRAP** will cause the loop to exit.

Line 91 is the destination of the **TRAP** statement, which closes the network connection, while printing two blank lines.

Line 100 resets the real time clock.

Line 110 sets up a loop which exits when PEEK(19) is equal to 30, which is approximately 3 minutes.

Line 120 jumps back to the TRAP statement on line 1, which re-starts the program, while taking care not to re-dimension the A$ variable.

WEBSRVR: A Simple Web Server

This example uses the TCP protocol to open a simple web server on port 8080. It does the absolute bare minimum of answering a connection, accepting a command, and then sending back a response to a browser, and should only be used as a minimal starting point, as a real HTTP server requires a lot more work, and is beyond the scope of this manual.

Once the program is waiting for a connection, you can connect with a browser to the IP address or host name of your FujiNet, and you should see the message "Hello from my Atari!" in a large bold font.

Example VIII.14. WEBSRVR Program Listing, in BASIC.

```
100 REM SIMPLE WEB SERVER
101 DIM A$(127)
110 OPEN #1,12,3,"N:TCP://:8080"
120 PRINT "WAITING..."
121 TRAP 250
130 STATUS #1,A
140 IF  NOT PEEK(748) THEN 130
150 PRINT "ACCEPTING..."
160 XIO ASC("A"),#1,12,3,"N:"
170 PRINT "READING HTTP COMMAND..."
171 STATUS #1,A
172 BW=PEEK(746)+256*PEEK(747)
173 IF NOT BW THEN 198
180 INPUT #1,A$
190 GOTO 171
199 PRINT "SENDING DOCUMENT..."
```

```
200 PRINT #1;"HTTP/1.1 200 OK"
210 PRINT #1;"Content-Type: text/html"
230 PRINT #1;""
240 PRINT #1;"<html>"
241 PRINT #1;"<head>"
242 PRINT #1;"<title>Atari Web Server</title>"
243 PRINT #1;"</head>"
244 PRINT #1;"<body>"
245 PRINT #1;" <h1>Hello from my Atari!</h1>"
246 PRINT #1;"</body>"
247 PRINT #1;"</html>"
250 XIO ASC("c"),#1,12,3,"N:"
260 GOTO 120
```

Line 100 is a simple remark indicating the program's function.

Line 101 defines a single string variable A$ which is 127 bytes long.

Line 110 opens a TCP listening port, at port 8080. Aux1 is set to 12, which means read/write, and aux2 is set to 3, which means to convert ATASCII EOL to and from an ASCII CR/LF pair.

Line 120 prints "WAITING..."

Line 121 sets a TRAP to line 250.

Line 130 Asks for the network channel status

Line 140 checks if the CONNECTED byte is set to 1, if so, we fall through, otherwise we loop back.

Line 150 prints "ACCEPTING..."

Line 160 uses XIO command "A" on the N: device, to accept the connection, taking care to use the same aux1 and aux2 parameters specified in OPEN.

Line 170 prints "READING HTTP COMMAND..."

Line 171 asks for network channel status

Line 172 sets BW to # of bytes waiting

Line 173 breaks out of the loop if no bytes waiting, to send the document.

Line 180 Reads the next line from the PC, we ignore it.

Line 190 loops back to 171, unless there isn't any data waiting.

Line 199 prints "SENDING DOCUMENT"

Lines 200-247 sends the HTTP response, and the document.

Line 250 closes the client connection

Line 260 returns to line 120

IX. FastBASIC

DMSC's FastBASIC is a very fast BASIC interpreter and compiler, able to not only leverage the unique hardware capabilities of the ATARI computers, but comes with FujiNet commands built into the language. These commands are implemented using the SIO interface, so the N: device handler is not required.

The FastBASIC package comes with an interpreter and compiler that run both on the native ATARI computer, but also cross-compilers which can run on any Windows or POSIX (e.g. MacOS and Linux) compliant system, making development fast and convenient.

The latest copy of FastBasic may be downloaded from GitHub: https://github.com/dmsc/fastbasic

Opening a Connection

The NOPEN statement is used to open a network connection.

```
NOPEN unit, aux1, aux2, url$
```

unit Selects the N: device to open, which can be from 1 to 8.

aux1 Sets the mode of the channel, 4 = read only (GET), 6 = directory (PROPFIND), 8 = write (PUT), 9 = append, 12 = read/write (GET), 13 = POST

aux2 Sets the translation mode, 0 = no translation, 1 = CR<->EOL, 2 = LF<->EOL, 3 = CR/LF<->EOL

url$ A string specifying the URL of the form: N:PROTOCOL://HOSTNAME:PORT/PATH...

Example IX.1. Opening a Network Connection, in FastBASIC.

```
NOPEN 1,12,2,"N:TELNET://BBS.FOZZTEXX.COM/"
```

Reading from Connection

The NGET statement can be used to read bytes from a connection, of a specified number of bytes, into a byte array.

```
NGET unit, adr, len
```

unit Selects the N: device to input, which can be from 1 to 8.

adr The address of the destination buffer in memory. Most commonly pointing to a byte array by way of the & operator.

len The exact number of bytes to read (1-65535)

Tip

`len` must be less than or equal to the number of bytes available in the receive buffer. Use **NSTATUS** followed by `DPEEK($02EA)` to get the number of bytes available.

Example IX.2. Reading from a Connection in FastBASIC

```
DIM BUF(256) BYTE
```

```
NSTATUS 1
BW = DPEEK($02EA)

' BECAUSE INTS ARE SIGNED, MUST
' TAKE VALUES > 32767 INTO
' ACCOUNT

IF BW > 256 OR BW < 000
  LN = 256
ELSE
  LN = BW

NGET 1,&BUF,LN
```

Writing to Connection

The **NPUT** statement can be used to write a length of bytes from a specific buffer to a network channel.

```
NPUT unit, adr, len
```

unit Selects the N: device to output, which can be from 1 to 8

adr The address of the source buffer in memory. Most commonly pointing to a byte array by way of the & operator.

len The exact number of bytes to write (1-65535)

Example IX.3. Writing to a connection, in FastBASIC

```
' GET KEY FROM KEYBOARD
' THEN SEND IT TO NETWORK CONNECTION

GET K
NPUT 1,&K,1
```

Getting Connection Status

The **NSTATUS** statement can be used to get the current status of a specific unit. The result is placed into the DVSTAT.

```
NSTATUS unit
```

unit Selects the N: device to acquire status, which can be from 1 to 8

The following table shows the contents of DVSTAT and what function should be used to read it.

Table IX.1. Contents of DVSTAT, in FastBASIC

Offset	Description
DPEEK($02EA)	Returns the number of bytes waiting in the receive buffer (1-65535)
PEEK($02EC)	1 = Channel connected, 0 = Disconnected
PEEK($02ED)	Returns the error code of the last operation.

Example IX.4. Obtaining Status, in FastBASIC

```
' Get current status
NSTATUS 1

' Put into useful variables
BW=DPEEK($02EA)
CONNECTED=DPEEK($02EC)
ERR=PEEK($02ED)
```

Closing a Connection

The **NCLOSE** statement will close a connection attached to a specific unit. If the given unit doesn't have an open connection, the NCLOSE statement will simply return to the main program.

```
NCLOSE unit
```

unit Selects the N: device to close, which can be from 1 to 8.

Example IX.5. Closing a connection, in FastBASIC

```
' Close connection on N: unit 1
NCLOSE 1
```

Caution

When a unit is opened as a listening socket, **NCLOSE** will close the listening socket. A delay of 2 minutes is imposed for any closed listening sockets, before they are available again for use. If you intend to re-use the listening socket to accept another connection, please refer to "**Closing the Server (Listening) Connection**"

Opening a Server (Listening) Connection

NOPEN can be used to create a listening socket on a specified port. This is used to accept connections from another computer.

Example IX.6. Opening a Server (Listening) Connection, in FastBASIC

```
' Open a TCP listening (server) socket on port 6502
' 0 = No Translation (e.g. Atari-to-Atari)

NOPEN 1,12,0,"N:TCP://:6502/"
```

Checking for a Client Connection

NSTATUS can be used to check the listening connection for a connection. PEEK($02EC) will return a 0 if there is no connection waiting, or a 1 if there is a connection waiting.

Example IX.7. Checking for a Server Connection, in FastBASIC

```
' NSTATUS must be run each connected check.
NSTATUS 1
CONNECTED=PEEK($02EC)
IF CONNECTED THEN
```

```
' ...
```

Accepting a Client Connection

Connections can be accepted by sending an 'A' command to the appropriate N: unit. The SIO command can be used to do this, and for convenience, can be wrapped into the following PROCedure:

```
PROC naccept unit, mode, trans
 SIO $71, unit, $41, $00, 0, $1f, 0, mode, trans
ENDPROC
```

unit Accept connection for unit #1-8

mode The channel mode to set, usually should match what was passed to NOPEN

trans The translation mode to set, usually should match what was passed to NOPEN

Example IX.8. Accepting a Client Connection in FastBASIC

```
naccept 1
```

Closing an Accepted Connection

The correct way to disconnect an accepted client connection is to use the following PROCedure, which sends a 'c' command to the appropriate network unit:

```
PROC ncloseclient unit
 SIO $71, unit, $63, $00, 0, $1f, 0, 0, 0
ENDPROC
```

unit Close any client connection for unit #1-8

Closing the Server (Listening) Connection

If a program is done with a listening socket, it should be closed with an NCLOSE statement. The example closes the server (listening) connection on unit 1:

Example IX.9. Closing the Server (Listening) Connection, in FastBASIC

```
NCLOSE 1
```

Example Programs

Note

The programs listed here, can be found in the FastBASIC samples/ directory.

NETCAT: A Simple Terminal Emulator

NETCAT is a program which acts like a simple dumb terminal emulator. It will read and display any information that appears on the network channel, while also checking for key presses and sending them out the same network channel. This will continue until either the host disconnects, or a network error occurs. In either case, the network channel is closed, and the program ends.

```
' -- #FUJINET NETCAT Example --

' Default unit # for connection
CONN=2
MODE=12
TRANS=0

' RX Buffer
DIM BUF(8192) BYTE

' Procedures ''''''''''''''''''''''''
PROC BANNER
 PRINT "** NETCAT IN FASTBASIC **"
 PRINT
ENDPROC

PROC GETCONN
 URL$="N:"
 INPUT "URL: ", C$
 URL$=+C$
 INPUT "TRANS (0=none, 1=CR, 2=LF, 3=CR/LF): ", TRANS
ENDPROC

PROC INTCLR
 POKE $D302, PEEK($D302) & 127
ENDPROC

PROC CONNECT
 PRINT "Connecting to:"
 PRINT URL$
 NOPEN CONN,MODE,TRANS,URL$
 NSTATUS CONN
ENDPROC

PROC IN
 ' Clear interrupt and get status
 @INTCLR
 NSTATUS CONN

 ' Check if we need to read data
 BW = DPEEK($02EA)
 WHILE BW
  ' Needs to handle the case where
  ' BW > 32767, giving negative.
  IF BW > 8192 OR BW < 000
   LN = 8192
  ELSE
   LN = BW
  ENDIF

  NGET CONN,&BUF, LN
  BPUT #0, &BUF, LN
  BW = BW - LN
 WEND
```

```
ENDPROC

PROC OUT
 GET K
 NPUT CONN, &K, 1
ENDPROC

PROC NC
 DO
  IF PEEK($D302) & 128
   @IN
  ENDIF

  IF PEEK($02EC) = 0
   PRINT "Disconnected."
   NCLOSE CONN
   EXIT
  ENDIF

  IF KEY() THEN @OUT
 LOOP
ENDPROC

''''''''''''''''''''''''
' Main Program

POKE 65,0 ' quiet SIO

@BANNER
@GETCONN
@CONNECT

IF SErr() <> 1
  NSTATUS CONN
  PRINT "Could not Make Connection"
  PRINT "ERROR- "; PEEK($02ED)
  NCLOSE CONN
ELSE
  PRINT "Connected!"
  @NC
ENDIF

POKE 65,3 ' noisy SIO
```

1 Variable definitions.

MASTODON: Show the newest Mastodon post

```
' A Mastodon Client in FastBASIC

' N: Unit to use
unit=8

' URL to Mastodon Server
```

```
url$="N:HTTPS://oldbytes.space/api/v1/timelines/public?limit=1"$9B

' QUERY string
query$=""

' QUERY result
DIM result(1024) BYTE

' JSON channel mode
JSON_MODE=1

' PROCEDURES ''''''''''''''''''''''''
PROC nprinterror
 NSTATUS unit
 PRINT "ERROR- "; PEEK($02ED)
ENDPROC

PROC nsetchannelmode mode
 SIO $71, unit, $FC, $00, 0, $1F, 0, 12, JSON_MODE
ENDPROC

PROC nparsejson
 SIO $71, unit, $50, $00, 0, $1f, 0, 12, 0
ENDPROC

PROC njsonquery
 SIO $71, unit, $51, $80, &query$+1, $1f, 256, 12, 0
ENDPROC

PROC showresult
 @njsonquery
 NSTATUS unit

 IF PEEK($02ED) >
  PRINT "Could not fetch query:"
  PRINT query$
  EXIT
 ENDIF

 BW=DPEEK($02EA)
 NGET unit, &result, BW
 BPUT #0, &result, BW
ENDPROC

PROC mastodon
 ' Open connection
 NOPEN unit, 12, 0, url$

 ' If not successful, then exit.
 IF SERR()<>1
  PRINT "Could not open connection."
  @nprinterror
  EXIT
 ENDIF
```

```
 ' Change channel mode to JSON
 @nsetchannelmode JSON_MODE

 ' Ask FujiNet to parse JSON
 @nparsejson

 ' If not successful, then exit.
 IF SErr()<>1
  PRINT "Could not parse JSON."
  @nprinterror
  EXIT
 ENDIF

 ' Show latest post
 query$="N:/0/account/display_name"$9B
 @showresult
 query$="N:/0/created_at"$9B
 @showresult
 query$="N:/0/content"$9B
 @showresult

 NCLOSE unit

 PRINT
 PRINT " ---- "
 PRINT

ENDPROC

' MAIN PROGRAM ''''''''''''''''''''''''
DO
 @mastodon
 PAUSE 1800
LOOP
```

X. ACTION!

ACTION! programs can access the FujiNet directly, via SIO, allowing for FujiNet programs to be used even when there is no N: handler loaded.

To make usage of FujiNet easier, a library called NIO.ACT is provided. It provides functions for opening, closing, reading, writing, and a few special features (such as JSON parsing and querying), and can be easily patterned to send any FujiNet command over SIO. The following sections show how to use the individual functions found in NIO.ACT.

Performing an SIO Operation

ACTION! can easily communicate directly with the FujiNet via SIO. This is accomplished by calling the operating system routine `SIOV` which has a known vector address of $E459. This routine reads its parameters from a fixed location in memory, starting at address $0300 called the Device Control Block (DCB). The procedure for using the FujiNet via `SIOV` is as follows:

1. Set `DDEVIC` to $71.
2. Set `DUNIT` to the N: device you wish to use. `ngetunit()` can help set this via a URL.
3. Set `DCOMND` to the command you wish to send.
4. Set `DSTATS` to indicate the direction of payload during transfer. $00 indicates there is no payload, $40 indicates to send from the FujiNet to the ATARI®, and $80 indicates that the payload goes from ATARI® to the FujiNet.
5. If `DSTATS` is $40 or $80, set `DBUF` to point to the correct buffer, otherwise simply set zero.
6. Set `DTIMLO` to a value larger than 0, to indicate # of seconds to wait for operation to complete.
7. If `DSTATS` is $40 or $80, set `DBYT` to the number of bytes to transfer from the buffer, otherwise simply set zero.
8. Set `DAUX1` to an appropriate value.
9. Set `DAUX2` to an appropriate value, or...
10. You can also set as a 16-bit value, if you define `DAUX` as a CARD.
11. Once everything is set, call `SIOV()`
12. Check `DSTATS` for error code.
13. If `DSTATS`=144, check `DVSTAT+3` (aka ERR) for error code.

The following code snippet defines the locations of the Device Control Block (DCB) and the function to call `SIOV`:

```
;
; DEVICE CONTROL BLOCK (DCB)
;
BYTE DDEVIC = $0300 ; Device #
BYTE DUNIT  = $0301 ; Unit #
```

```
BYTE DCOMND = $0302 ; Command
BYTE DSTATS = $0303 ; <-> and error
CARD DBUF   = $0304 ; buffer
BYTE DTIMLO = $0306 ; timeout secs
BYTE DUNUSE = $0307 ; reserved
CARD DBYT   = $0308 ; pyld byte len
CARD DAUX   = $030A ; daux1/daux2
BYTE DAUX1  = $030A ; daux1
BYTE DAUX2  = $030B ; daux2

;
; PROC to call SIO Vector (SIOV)
;
PROC siov=$E459()
```

Setting DUNIT via a URL

The function `ngetunit()` will parse the URL pointed to by `ds` to get the requested unit number. It will parse N: as unit 0:

```
;
; Get the unit number from devicespec
;
BYTE FUNC ngetunit(BYTE ARRAY ds)
    BYTE unit=1

    IF ds(2)=': THEN
        unit=1
    ELSEIF ds(3)=': THEN
        unit=ds(2)-$30
    ELSE
        unit=1
    FI

RETURN (unit)
```

Getting the error code

The procedure `geterror()` can be used to derive the error code of the completed operation.

```
BYTE EXTERR = $02ED
BYTE DSTATS = $0303

;
; Return error of last NIO operation.
; If SIO error = 144, then a status
; is done, and the extended err is
; returned.
;
; @param devicespec N: devicespec
; @return error 1=successful
;
BYTE FUNC geterror(BYTE ARRAY ds)
```

```
  BYTE errno

  IF DSTATS=144 THEN
    nstatus(ds)
    errno=EXTERR
  ELSE
    errno=DSTATS
  FI

RETURN (errno)
```

Opening a Connection

The following function `nopen()` can be used to open a network connection with a given protocol:

```
;
; Open the N: device pointed to by
; devicespec.
;
; @param devicespec N: devicespec
; @param trans - translation mode
;   0=NONE, 1=CR, 2=LF, 3=CR/LF
; @return error, 1=successful.
;
BYTE FUNC nopen(BYTE ARRAY ds, BYTE t)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'O    ; OPEN
  DSTATS = $80   ; Write to fujinet
  DBUF   = ds    ; send devicespec
  DTIMLO = $1F   ; 32 second timeout
  DBYT   = 256   ; 256 byte payload
  DAUX1  = 12    ; R/W
  DAUX2  = t     ; translation
  siov();

RETURN (geterror(ds))
```

ds A string containing the device spec to open, such as:

```
N:HTTPS://WWW.GNU.ORG/licenses/gpl-3.0.txt
```

t The translation mode to use where:

Table X.1. nopen Translation Values

t	Description
0	No Translation of end of line (EOL) characters.
1	Translate EOL character to/from CR
2	Translate EOL character to/from LF
3	Translate EOL character to/from CR/LF pair

Tip

the `nenableproc()` procedure can be used after `nopen()` to enable an interrupt which will set the variable trip to TRUE when traffic is available to read.

Example X.1. Opening a Connection in ACTION! Using nopen()

```
' 2 = use UNIX line endings.
nopen("N:HTTP://WWW.GNU.ORG/licenses/gpl-3.0",2);
```

PROCEED Interrupt Handler

The FujiNet signals incoming traffic via an interrupt on the PROCEED pin. This interrupt is edge triggered. The ATARI operating system has an interrupt vector called `VPRCED`, which is called from the operating system's IRQ handler. Using interrupts can drastically improve performance of interactive network applications by minimizing the number of polls needed to determine how many bytes are waiting in FujiNet's receive buffer.

Not all programs need this functionality, so it may be omitted in cases where constant polling of the receive buffer isn't required. Such applications include those which use the HTTP protocol.

To react to this interrupt, a handler needs to be placed in memory, and VPRCED's address vector needs to point to it. The following is a sample interrupt handler which sets a variable called trip to 1 whenever the PROCEED pin is changed in response to data being in the receive buffer:

```
;
; Interrupt handler
;
; A9 01    LDA #$01
; 8D XX XX STA trip
; 68       PLA
; 40       RTI
;
PROC ninterrupt_handler=*()
[$A9$01$8D trip $68$40]
```

The PROCEED interrupt will trigger if any active network devices have data in their respective receive buffers.

Enabling PROCEED Interrupt

If you choose to use the PROCEED interrupt functionality, you should use this procedure to enable the interrupt safely and to properly reset the `trip` variable to zero:

```
;
; PROCEED interrupt vars
;
CARD VPRCEDSAVE     ; Save vector for vprced
CARD VPRCED = $0202 ; Proceed vector
BYTE PACTL  = $D302 ; PIA Control for Proceed
BYTE trip           ; Trip FLAG.
```

```
;
; Enable Interrupt handler.
;
PROC nenableproc()

    trip=0
    VPRCEDSAVE=VPRCED
    VPRCED=ninterrupt_handler
    PACTL = PACTL % 1

RETURN
```

Important

once `nenableproc()` is used, `ndisableproc()` must be called, either on closing of the network connection, or program. Otherwise, unpredictable system behavior may result! See Disabling PROCEED Interrupt.

Reading From a Connection

The following function, called `nread()`, can read from a network connection that has been opened by the `nopen()` function:

```
;
; Read len bytes from N: device
; pointed to by devicespec.
;
; @param devicespec N: devicespec
; @param buf The dest buffer
; @param len # of bytes to read
; @return error 1=successful
;
BYTE FUNC nread(BYTE ARRAY ds, BYTE ARRAY buf, CARD len)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'R    ; READ
  DSTATS = $40   ; Atari<-Payload
  DBUF   = buf   ; send devicespec
  DTIMLO = $1F   ; 32 second timeout
  DBYT   = len   ; No payload
  DAUX   = len
  siov();

RETURN (geterror(ds))
```

ds — The Device specification previously passed to `nopen()`, used to derive the network device unit number.

buf — The byte array to place read characters into. Must be able to hold at least `len` number of bytes.

len — The number of bytes to read (1-65535).

Warning

`buf` must be able to hold the number of bytes requested by `len`. `Len` must be less than, or equal to the number of bytes available in the receive buffer. Use `nstatus()` to determine the number of bytes available in the receive buffer.

For example, to read data from an open connection, while making sure to only read as much as is available, but not more than the target buffer can allow:

Example X.2. Reading from a Connection, in ACTION! using nread()

```
CARD bw = $02EA  ; DVSTAT
BYTE err

err = nstatus(url)

if err <> 1 then
    return
fi

if bw = 0 then
    return
fi

if bw > 4096 then
    bw = 4096
fi

nread(url,buf,bw)
```

Writing To a Connection

The following function, called `nwrite()`, can write to a network connection that has been opened by the `nopen()` function.

```
;
; Write len bytes to N: device
; pointed to by devicespec.
;
; @param devicespec N: devicespec
; @param buf The src buffer
; @param len # of bytes to read
; @return error 1=successful
;
BYTE FUNC nwrite(BYTE ARRAY ds, BYTE ARRAY buf, CARD len)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'W    ; WRITE
  DSTATS = $80   ; Payload->FujiNet
  DBUF   = buf   ; send devicespec
  DTIMLO = $1F   ; 32 second timeout
  DBYT   = len   ; No payload
```

```
  DAUX   = len
  siov();

RETURN (geterror(ds))
```

ds The Device specification previously passed to `nopen()`, used to derive the network device unit number.

buf The byte array to read characters from. Must have at `len` number of bytes.

len The number of bytes to write. (1-65535).

Warning

`buf` must contain at least the number of bytes specified by `len`.

For example, to write data to an open connection:

Example X.3. Writing to a Connection, in ACTION! using nwrite()

```
nwrite("Hello",5);
```

Getting Connection Status

The following function, called nstatus(), can return status information in a connection into the system variable DVSTAT, which exists at location $02EA, which includes:

Table X.2. Return Values in DVSTAT for nstatus()

Variable	Description
CARD BW=$02EA	The Number of bytes waiting in a connection's receive buffer.
BYTE CONNECTED=$02EC	Is the connection active? 1 = TRUE, 0 = FALSE
BYTE ERROR=$02ED	The error from the previous network operation.

```
CARD BW=$02EA
BYTE CONNECTED=$02EC
BYTE ERROR=$02ED

;
; Get status of last NIO operation,
; Return in DVSTAT
;
; @param devicespec N: devicespec
;
PROC nstatus(BYTE ARRAY ds)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'S     ; STATUS
  DSTATS = $40    ; Payload to Atari
  DBUF   = $02EA  ; status buffer
  DTIMLO = $1F    ; 32 second timeout
  DBYT   = 4      ; 4 byte payload
```

```
  DAUX1  = 0      ; R/W
  DAUX2  = 0      ; translation
  siov()

RETURN
```

ds The Device specification previously passed to `nopen()`, used to derive the network device unit number.

For example, to check if an open connection has been disconnected:

Example X.4. Getting Connection Status in ACTION! using nstatus()

```
BYTE CONNECTED=$02EC

nstatus("N:TELNET://BBS.FOZZTEXX.COM")

if (CONNECTED=0)
    EXIT()
fi
```

Disabling PROCEED Interrupt

If `nenableproc()` was used after opening a network connection, the following procedure `ndisableproc()` must be used to disable the interrupt before the program ends.

```
CARD VPRCEDSAVE

;
; Disable Interrupt handler.
;
PROC ndisableproc()

    trip=0
    VPRCED=VPRCEDSAVE

RETURN
```

Important

If `nenableproc()` is used, Failure to call `ndisableproc()` before program exit will result in unpredictable system operation!

Closing a Connection

The following function, called nclose(), will close a connection previously opened with nopen():

```
;
; Close the N: device pointed to by
; devicespec.
;
; @param devicespec N: devicespec
; @return error, 1=successful.
;
```

```
BYTE FUNC nclose(BYTE ARRAY ds)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'C    ; CLOSE
  DSTATS = $00   ; No Payload
  DBUF   = 0     ; send devicespec
  DTIMLO = $1F   ; 32 second timeout
  DBYT   = 0     ; No payload
  DAUX1  = 0
  DAUX2  = 0
  siov();

RETURN (geterror(ds))
```

Opening a Server (Listening) Connection

If a host name is not specified in the URL passed to `nopen()`, FujiNet will attempt to open a server connection, which listens for connection attempts to a specific port.

```
' Open a TCP listening connection, on port 6502. No translation.
nopen("N:TCP://:6502/",0)
```

Checking for a Client Connection

Once a server connection is opened, `nstatus()` can be used to populate the CONNECTED variable (which is located at DVSTAT+2), to determine if a connection is waiting to be accepted.

```
BYTE CONNECTED=$02EC
BYTE ARRAY URL="N:TCP://:6502"

nstatus(url)

IF CONNECTED=1 THEN
    naccept(url); ' See Accepting a Client Connection
FI
```

Accepting a Client Connection

The following procedure, called `naccept()` can be used to accept a client connection. The connection can immediately be treated as if it were a client connection:

```
;
; Accept the Client Connection
;
; @param ds devicespec N: devicespec
;
BYTE FUNC naccept(BYTE ARRAY ds)
  DDEVIC=$71
  DUNIT=ngetunit(ds)
  DCOMND='A  ; Accept
  DSTATS=$00 ; No Payload
  DBUF=0     ; No Payload
```

```
  DTIMLO=$1F ; 32 second timeout
  DBYT=0     ; No payload
  DAUX1=0
  DAUX2=0
  siov()

RETURN (geterror(ds))
```

Closing an Accepted Client Connection

The following procedure can be used to close a client connection, while still keeping the listener connection open for another host to connect:

```
;
; Close the Client Connection
; Free the listening connection
; For further use.
;
; @param ds devicespec N: devicespec
;
BYTE FUNC ncloseclient(BYTE ARRAY ds)
  DDEVIC=$71
  DUNIT=ngetunit(ds)
  DCOMND='c  ; client close, lower case c
  DSTATS=$00 ; No Payload
  DBUF=0     ; No Payload
  DTIMLO=$1F ; 32 second timeout
  DBYT=0     ; No payload
  DAUX1=0
  DAUX2=0
  siov()

RETURN (geterror(ds))
```

Closing the Server (Listening) Connection

Closing the server listening connection, should be done when the program no longer needs to listen as a server for remote connections. nclose() is used for this purpose:

```
BYTE ARRAY url="N:TCP://:6502"

nclose(url)
```

NIO.ACT Library Listing

NIO.ACT is an ACTION! Library that exposes FujiNet functions via SIO calls. It also provides an interrupt handler routine that improves performance by minimizing status polling.

This library may be fetched directly on-line via the URL: https://raw.githubusercontent.com/FujiNetWIFI/fujinet-apps/master/netcat-action/NIO.ACT

```
;
; #FujiNet Network I/O Library
```

```
; For ACTION!
;
; Author: Thomas Cherryhomes
;  <thom.cherryhomes@gmail.com>
;
; These routines call the #Fujinet
; directly from SIO, and thus do
; not need the N: handler (NDEV)
;

MODULE

;
; PROCEED interrupt vars
;
CARD VPRCEDSAVE     ; Save vector for vprced
CARD VPRCED = $0202 ; Proceed vector
BYTE PACTL  = $D302 ; PIA Control for Proceed
BYTE trip           ; Trip FLAG.

;
; DVSTAT (Status)
;
BYTE DVSTAT = $02EA ; PTR TO DVSTAT
BYTE EXTERR = $02ED ; DVSTAT+3

;
; DEVICE CONTROL BLOCK (DCB)
;
BYTE DDEVIC = $0300 ; Device #
BYTE DUNIT  = $0301 ; Unit #
BYTE DCOMND = $0302 ; Command
BYTE DSTATS = $0303 ; <-> and error
CARD DBUF   = $0304 ; buffer
BYTE DTIMLO = $0306 ; timeout secs
BYTE DUNUSE = $0307 ; reserved
CARD DBYT   = $0308 ; pyld byte len
CARD DAUX   = $030A ; daux1/daux2
BYTE DAUX1  = $030A ; daux1
BYTE DAUX2  = $030B ; daux2

;
; Interrupt handler
;
; A9 01    LDA #$01
; 8D XX XX STA trip
; 68       PLA
; 40       RTI
;
PROC ninterrupt_handler=*()
[$A9$01$8D trip $68$40]

;
; Enable Interrupt handler.
```

```
;
PROC nenableproc()

    trip=0
    VPRCEDSAVE=VPRCED
    VPRCED=ninterrupt_handler
    PACTL = PACTL % 1

RETURN

;
; Disable Interrupt handler.
;
PROC ndisableproc()

    trip=0
    VPRCED=VPRCEDSAVE

RETURN

;
; PROC to call SIO Vector (SIOV)
;
PROC siov=$E459()

;
; Get the unit number from devicespec
;
BYTE FUNC ngetunit(BYTE ARRAY ds)
    BYTE unit=1

    IF ds(2)=': THEN
        unit=1
    ELSEIF ds(3)=': THEN
        unit=ds(2)-$30
    ELSE
        unit=1
    FI

RETURN (unit)

;
; Get status of last NIO operation,
; Return in DVSTAT
;
; @param devicespec N: devicespec
;
PROC nstatus(BYTE ARRAY ds)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'S     ; STATUS
  DSTATS = $40    ; Payload to Atari
  DBUF   = $02EA  ; status buffer
```

```
  DTIMLO = $1F    ; 32 second timeout
  DBYT   = 4      ; 4 byte payload
  DAUX1  = 0      ; R/W
  DAUX2  = 0      ; translation
  siov()

RETURN

;
; Return error of last NIO operation.
; If SIO error = 144, then a status
; is done, and the extended err is
; returned.
;
; @param devicespec N: devicespec
; @return error 1=successful
;
BYTE FUNC geterror(BYTE ARRAY ds)
  BYTE errno

  IF DSTATS=144 THEN
    nstatus(ds)
    errno=EXTERR
  ELSE
    errno=DSTATS
  FI

RETURN (errno)

;
; Open the N: device pointed to by
; devicespec.
;
; @param devicespec N: devicespec
; @param trans - translation mode
;   0=NONE, 1=CR, 2=LF, 3=CR/LF
; @return error, 1=successful.
;
BYTE FUNC nopen(BYTE ARRAY ds, BYTE t)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'O    ; OPEN
  DSTATS = $80   ; Write to fujinet
  DBUF   = ds    ; send devicespec
  DTIMLO = $1F   ; 32 second timeout
  DBYT   = 256   ; 256 byte payload
  DAUX1  = 12    ; R/W
  DAUX2  = t     ; translation
  siov();

  IF DSTATS=1 THEN
      nenableproc()
  FI
```

```
RETURN (geterror(ds))

;
; Close the N: device pointed to by
; devicespec.
;
; @param devicespec N: devicespec
; @return error, 1=successful.
;
BYTE FUNC nclose(BYTE ARRAY ds)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'C    ; CLOSE
  DSTATS = $00   ; No Payload
  DBUF   = 0     ; send devicespec
  DTIMLO = $1F   ; 32 second timeout
  DBYT   = 0     ; No payload
  DAUX1  = 0
  DAUX2  = 0
  siov();

  ndisableproc()

RETURN (geterror(ds))

;
; Read len bytes from N: device
; pointed to by devicespec.
;
; @param devicespec N: devicespec
; @param buf The dest buffer
; @param len # of bytes to read
; @return error 1=successful
;
BYTE FUNC nread(BYTE ARRAY ds, BYTE ARRAY buf, CARD len)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'R    ; READ
  DSTATS = $40   ; Atari<-Payload
  DBUF   = buf   ; send devicespec
  DTIMLO = $1F   ; 32 second timeout
  DBYT   = len   ; No payload
  DAUX   = len
  siov();

RETURN (geterror(ds))

;
; Write len bytes to N: device
; pointed to by devicespec.
;
```

```
; @param devicespec N: devicespec
; @param buf The src buffer
; @param len # of bytes to read
; @return error 1=successful
;
BYTE FUNC nwrite(BYTE ARRAY ds, BYTE ARRAY buf, CARD len)

  DDEVIC = $71
  DUNIT  = ngetunit(ds)
  DCOMND = 'W    ; WRITE
  DSTATS = $80   ; Payload->FujiNet
  DBUF   = buf   ; send devicespec
  DTIMLO = $1F   ; 32 second timeout
  DBYT   = len   ; No payload
  DAUX   = len
  siov();

RETURN (geterror(ds))

; Get FujiNet Status
;
; Author: Michael Goroll
;  <atari@goroll.net>
;
PROC fstatus(BYTE ARRAY buf)

DDEVIC = $70
DUNIT  = $01
DCOMND = $E8
DSTATS = $40    ; Payload to Atari
DBUF   = buf    ; status buffer
DTIMLO = $0F    ; 8 second timeout
DBYT   = $88    ; 139 byte payload
DAUX1  = 0      ; R/W
DAUX2  = 0      ; translation
siov()

RETURN
```

Example Programs

The following programs are listed here for easy reference.

They are loadable via TNFS on apps.irata.online, in the directory: /Atari_8-bit/Internet/ as:

- Netcat (Action).atr
- Mastodon (Action).atr

NETCAT: A Simple Terminal Emulator

NETCAT is a program which acts like a simple dumb terminal emulator. It will read and display any information that appears on the network channel, while also checking for key presses and

sending them out the same network channel. This will continue until either the host disconnects, or a network error occurs. In either case, the network channel is closed, and the program ends.

Example X.5. NETCAT ACTION! Listing

```
;
; A simple netcat program
; to show how to do basic network
; input and output.
;
; Author: Thomas Cherryhomes
;  <thom.cherryhomes@gmail.com>
;

INCLUDE "D2:SYS.ACT"
INCLUDE "D2:NIO.ACT"

MODULE

CARD BYTESWAITING=$02EA     ; # of bytes waiting
BYTE KP=$02FC               ; Key pressed?
BYTE ARRAY devicespec(256)  ; DeviceSpec
BYTE trans                  ; translation mode
BYTE localEcho              ; local echo off/on?
BYTE running                ; is program running?
BYTE ARRAY rxbuf(8192)      ; receive buffer

DEFINE KEYBOARD_IOCB="2"
DEFINE TRUE="1"
DEFINE FALSE="0"

;
; Prompt for URL
;
PROC getURL()
       PrintE("NETCAT--ENTER DEVICE SPEC?")
       InputS(devicespec)
       PutE()

       PrintE("TRANS--0=NONE, 1=CR, 2=LF, 3=CR/LF?")
       trans=InputB()
       PutE()

       PrintE("LOCAL ECHO--0=NO, 1=YES?")
       localEcho=InputB()
       PutE()
RETURN

;
; Handle nc output
;
PROC ncoutput()
       BYTE ARRAY ch(1)
       BYTE err
```

```
        IF KP=$FF THEN
            RETURN
        FI

        ch(0)=GetD(KEYBOARD_IOCB)

        IF localEcho=1 THEN
            Put(ch(0))
        FI

        err=nwrite(devicespec,ch,1)

        IF err<>1 THEN
            Print("Write Error: ")
            PrintB(err)
            running=FALSE
        FI

RETURN

;
; Handle nc input
;
PROC ncinput()
       BYTE err
       CARD I

       IF trip=0 THEN
           RETURN
       FI

       nstatus()

       IF EXTERR=136 THEN
          PrintE("Disconnected.")
          running=FALSE
          RETURN
       FI

       IF BYTESWAITING=0 THEN
          RETURN
       FI

       IF BYTESWAITING>8192 THEN
          BYTESWAITING=8192
       FI

       ; Do the network read.
       err=nread(devicespec,rxbuf,BYTESWAITING)

       IF err<>1 THEN
           Print("Read Error: ")
           PrintB(err)
```

```
            running=FALSE
            RETURN
        FI

        ; Drain/display rx buffer
        FOR I=0 TO BYTESWAITING-1
        DO
            Put(rxbuf(I))
        OD

        ; Done, reset interrupt
        trip=0
        PACTL=PACTL%1

RETURN

;
; The main Netcat function
;
PROC nc()
        BYTE err

        err=nopen(deviceSpec,trans)

        IF err<>1 THEN
            Print("Open Error: ")
            PrintB(err)
            RETURN
        FI

        ; flag program as running.
        running=1

        ; Open keyboard
        Open(KEYBOARD_IOCB,"K:",4,0)

        WHILE running = TRUE
          DO
          ncoutput()
          ncinput()
          OD

        PrintE("Bye.")

        ; Clean up
        Close(KEYBOARD_IOCB) ; close kybd
        nclose(deviceSpec)

RETURN

;
; Main entrypoint
;
PROC main()
```

```
        getURL()
        nc()
RETURN
```

MASTODON: Show the Newest Mastodon Post

```
; A SIMPLE MASTODON CLIENT
; SHOWS THE LATEST POST
; FROM OLDBYTES.SPACE

INCLUDE "D1:SYS.ACT"
INCLUDE "D1:NIO.ACT"

MODULE

BYTE ARRAY URL(256)
BYTE ARRAY QS(256)
BYTE CONSOL=$D01F
BYTE CLOK0=$12
BYTE CLOK1=$13
BYTE CLOK2=$14
CARD BW=$02EA

; PRINT RESULT OF JSON QUERY
; _qs = Query String
PROC PRINTJ(BYTE ARRAY _qs)
    BYTE ERR
    BYTE ARRAY OUT(4096)
    CARD X

    SCOPY(QS,_qs)

    ; get length of string
    ; append EOL
    X=QS(0)
    QS(X+1)=155

    ; perform query
    ERR = NQUERYJSON(URL,QS)

    IF ERR<>1 THEN
       PRINT("QUERY ERROR: ")
       PRINTB(ERR)
       RETURN
    FI

    ; Get # of bytes waiting...
    NSTATUS(URL)

    IF ERR<>1 THEN
       PRINT("STATUS ERROR: ");
       PRINTBE(ERR)
       RETURN
```

```
    FI

    ; Read result into OUT
    ERR = NREAD(URL,OUT,BW)

    IF ERR<>1 THEN
       PRINT("READ ERROR: ")
       PRINTBE(ERR)
       RETURN
    FI

    ; Display it, one byte at a time
    FOR X=0 TO BW-1
    DO
        PUT(OUT(X))
    OD

RETURN

PROC FOOTER()
    PRINTE("")
    PRINTE("-------------------------------------")
    PRINTE("")
RETURN

PROC BANNER()
    FOOTER()
    PRINTE("LATEST POST FROM:")
    PRINTE(" OLDBYTES.SPACE ")
    FOOTER()
RETURN

PROC SHOW_POST()
    BYTE ERR

    SCOPY(URL,"N:HTTPS://oldbytes.space/api/v1/timelines/public?limit=1")

    ERR=NOPEN(URL,0)

    IF ERR<>1 THEN
       PRINT("OPEN ERROR: ")
       PRINTBE(ERR)
       RETURN
    FI

    NCHANNELMODE(URL,1)

    ERR=NPARSEJSON(URL)

    IF ERR<>1 THEN
       PRINT("PARSE ERROR: ")
       PRINTBE(ERR)
       RETURN
    FI
```

```
    PRINTJ("N:/0/account/display_name")
    PRINTJ("N:/0/created_at");
    PRINTJ("N:/0/content");

    NCLOSE(URL)

RETURN

PROC MAIN()

    BANNER()
    SHOW_POST()
    FOOTER()

RETURN
```

XI. Assembler

XII. FORTH

XIII. C

(using CC65)

XIV. Pascal

(using Mad Pascal)

Part III. REFERENCE

TABLE OF CONTENTS

XV. FujiNet Configuration Tools

FCONFIG

Show Network Configuration

FCONFIG

Description

FCONFIG reads the adapter configuration from FujiNet and displays it.

It currently displays:

- Wireless Network SSID
- Host name
- Local IP address
- Gateway IP address
- DNS IP address
- Netmask
- Hardware (MAC) address
- BSSID
- **FCONFIG** Version
- FujiNet Firmware Version

Parameters

This command has no parameters.

Messages

FCONFIG displays output similar to the following:

```
D1:FCONFIG

          SSID: Dummy Cafe
      Hostname: TMA-2
    IP Address: 127.0.0.1
Gateway Address: 0.0.0.0
   DNS Address: 1.1.1.1
       Netmask: 255.0.0.0
   MAC Address: D0:1C:ED:C0:FF:EE
         BSSID: D0:1C:ED:C0:FF:EE
```

```
FCONFIG Version: 0.1.7643858
  Fuji Firmware: v1.2

D1:
```

Returns

Returns 0 on success. Otherwise the SIO error code is returned.

FEJECT

Eject disk image in specified slot

FEJECT *<ds#>*

Description

Ejects the disk image specified by the device slot. A notification is displayed if there is no disk image in the specified slot.

Parameters

ds# The device slot # (1-8)

Messages

On success, the following is displayed:

```
Disk D3: ejected.
```

Returns

Error code of 0 on success, 1 on invalid parameters, otherwise the SIO error code is returned.

FHOST

Edit or clear a host slot

FHOST *<hs#>[,hostname]*

Description

If a host name is specified, it will change the host name of the specified host slot, otherwise the host slot is cleared.

Parameters

<hs#> Required. The host slot # (1-8)

[hostname] The host name to set.

Examples

Example XV.1. Setting Host Slot #4 with FHOST

To set host slot #4 to 192.168.1.8:

```
FHOST 4,192.168.1.8
```

Example XV.2. Clearing Host Slot #4 with FHOST

To clear the host name in slot #4:

```
FHOST 4
```

Messages:

If a host entry is changed:

```
Host Slot #4 changed to:
192.168.1.8
```

If a host entry is cleared:

```
Host slot #4 cleared.
```

Returns

Error code of 0 on success, 1 on invalid parameters, otherwise the SIO error code is returned.

FINFO

Show disk info for given Device Slot

```
FINFO <ds>
```

Description

Shows the disk geometry of a disk image inserted into the specified device slot.

Parameters

<ds#> Required. The device slot # (1-8)

Example

Example XV.3. Showing Disk Info in Slot 1 with FINFO

```
D1:FINFO 1

Number of Tracks: 40
Sectors per Track: 18
```

```
Number of Sides: 1
Sector Size: 256

Disk Type: 180K SS/DD
```

Returns

Returns 0 on success. Otherwise the SIO error code is returned.

FLD

List Device Slots

FLD

Description

Displays the images mounted in each of the possible device slots D1: to D8:. Each entry shows the following information:

- Device Slot
- Host Slot
- File Mode (R = Read, W = Write)
- File Name

Example

Example XV.4. FLD Output

```
D1:FLD

D1: (3) (R) OSS DOS XL 2.30p.atr
D2: (3) (R) fnc-tools-doc.atr
D3: Empty
D4: Empty
D5: Empty
D6: Empty
D7: Empty
D8: Empty
```

This example shows two disk images, D1 and D2. Both disk images are mounted from host slot (3). Both are mounted (R)ead only.

Returns

Error code of 0 on success, otherwise the SIO error is returned.

FLH

List Host Slots

FLH

Description

Displays the host names in each of the eight host slots. "Empty" is displayed, if a host slot is not being used.

Example

Example XV.5. Listing Host Slots with FLH

```
D1:FLH

1: SD
2: tnfs.fujinet.online
3: apps.irata.online
4: fujinet.atari8bit.net
5: fujinet.pl
6: tma-2
7: Empty
8: ec.tnfs.io
```

Returns

Error code of 0, on success. Otherwise, the SIO error code is returned.

FLS

List files on a Host Slot

`FLS <hs#>,<path>`

Description

Lists files present on the server at the specified host slot, and path.

Parameters

`<hs#>` Required. The Host Slot to list. (1-8)

`<path>` Required. The path to list. Use '/' for the root directory.

Example

Example XV.6. Listing a server directory, using FLS

```
D1:FLH

1: SD
2: tnfs.fujinet.online
3: apps.irata.online
4: fujinet.atari8bit.net
```

```
5: fujinet.pl
6: tma-2
7: Empty
8: ec.tnfs.io

D1:FLS 2,/
ADAM/
APPLEII/
ATARI/
CBM/
TNFS-links/
```

Returns

Error code of 0 on success, 1 on invalid parameters, otherwise the SIO error code is returned.

FMALL

Mount All Device Slots

FMALL

Description

FMALL reads the currently saved device slots and mounts them all.

Returns

Returns 0 on success, otherwise the SIO error is returned.

FMOUNT

Mount a Disk Image

FMOUNT *<ds#>,<hs#>,<R|W>,<path>*

Description

Mounts a disk image from the server specified by host slot, to the specified device slot, with the appropriate mode (read or write).

Parameters

ds# The Device Slot # to use (1-8)

hs# The Host Slot # to use (1-8)

R|W The image mode. R = Read Only, W = Read/Write

Example

To mount `MyDOS 4.53.atr` from host slot 6, into Drive Slot 2:

Example XV.7. Mounting a disk with FMOUNT

```
D1:FLH

1: SD
2: tnfs.fujinet.online
3: apps.irata.online
4: fujinet.atari8bit.net
5: fujinet.pl
6: tma-2
7: Empty
8: ec.tnfs.io


D1:FLS 6,/
fastbasic.atr
Mastodon (ACTION!).atr
MASTODON.ACT
netcat-action.atr
NIO.ACT
Web_Server_example.atr

D1:FMOUNT 2,3,R,netcat-action.atr
D2: (3) (R) netcat-action.atr
```

Returns

Error code of 0 on success, 1 on invalid parameters, else the SIO error code is returned.

FNET

Connect to SSID with Password

```
FNET [ssid] [password]
```

Description

Connects to the specified wireless network, with the specified password. Password is optinal. If SSID isn't specified, then the last successful SSID is used.

Parameters

[ssid]	Optional. Specifies SSID to connect FujiNet to.
[password]	Optional. Specifies password to use.

Returns

...

FNEW

Create a New Disk Image

FNEW *<ds#>,<hs#>,<t>,<filename>*

Description

Creates a new disk image on the desired host, with the requested geometry and mounts it into the requested device slot, in read/write mode.

The resulting disk image is blank, and has no file system, therefore it must be formatted by the desired operating system, before use.

Parameters

ds# Device slot # to use (1-8)

hs# Host Slot # to place new disk image in (1-8)

t Disk image type, where t can be:

Table XV.1. FNEW Disk types

t	Description
1	90K (SS/SD)
2	140K (SS/ED)
3	180K (SS/DD)
4	360K (DS/DD)
5	720K (DS/QD)
6	1440K (DS/HD)
1-65535:128|256	Custom, specify # of sectors and sector size.

Messages

If the disk image is successfully mounted, the new device slot information will be displayed.

```
D2: (3) (W) newdisk.atr
```

Otherwise, an error message will be displayed.

Examples

Example XV.8. Creating a 90K disk with FNEW

This example places a 90K disk in device slot 3 (D3:), and places the image file on slot 1, which as seen by the output of **FLH**, is the SD card slot.

```
D1:FLH

1: SD
2: tnfs.fujinet.online
3: apps.irata.online
4: fujinet.atari8bit.net
```

```
5: fujinet.pl
6: Empty
7: Empty
8: Empty

D1:FNEW 3,1,1,BLAH1.ATR

CREATING DISK
D3: (1) (W) BLAH1.ATR
```

Example XV.9. Creating a 16MB disk with FNEW

This example places a 16 megabyte disk, made up of 65535 sectors, 256 bytes each, in device slot 4 (D4:), and places the image file on slot 1, assuming the same output of FLH as above, is the SD card slot.

```
D1:FNEW 3,1,65535:256,16MEG.ATR
```

Returns

Error code of 0 on success, 1 on invalid parameters, otherwise the SIO error code is returned.

FRESET

Reset FujiNet and do a Cold Start

FRESET

Description

This command tells the FujiNet to reset itself. After waiting a few moments, the ATARI® is cold started.

FSCAN

Scan for Wireless Networks

FSCAN

Description

Tells FujiNet to scan for wireless networks, and display their names. You can use **FNET** to connect to an SSID returned by **FSCAN**.

Examples

Example XV.10. Example run of FSCAN

```
D1:FSCAN

* Cherryhomes
* TomsNetwork
```

```
* SomebodyElse
* AndAnother1234
```

Returns

Error code of 0 on success, 1 on invalid parameters, otherwise the SIO error code is returned.

NCD

Change N: Prefix

NCD *[Nx:]<path>*

Description

This command is used to set a path which is always prepended before user input, such as to change the currently active directory on a network file system, or to set a consistent host name for TCP or UDP connections.

Parameters

[Nx:] Set the prefix for the specified network device (N1: to N8:)

<path> Set the new prefix, according to the following rules:

- If *<path>* contains a protocol, such as N:HTTP:// then the entire prefix is re-set absolutely.
- If *<path>* is *Nx:/* then the prefix is truncated to the end of the current host name, so `N:TNFS://APPS.IRATA.ONLINE/ATARI_8-BIT/Games` becomes `N:TNFS://APPS.IRATA.ONLINE/`
- If *<path>* is *Nx:* then the prefix is cleared completely.
- If *Nx:* is not specified, then `N1:` is assumed.
- If no protocol is specified, and `/` is the first character, then the path after the host name is set to the new value, so given a current prefix of `N:TNFS://APPS.IRATA.ONLINE/DOS/` and `N:/Games` is given, then the new prefix will be `N:TNFS://APPS.IRATA.ONLINE/Games/`
- If no protocol is specified, and the first character is alphanumeric, then the given value is appended to the current prefix. So, if the prefix is currently `N:TNFS://APPS.IRATA.ONLINE/` and `Games/` is given, the new prefix will be `N:TNFS://APPS.IRATA.ONLINE/Games/`
- If two periods are specified, then the prefix will have one path component removed, usually leading to the parent directory.

Examples

Example XV.11. Setting Network Prefix with NCD

```
D1:NCD N:TNFS://APPS.IRATA.ONLINE/
```

```
D1:NPWD
TNFS://APPS.IRATA.ONLINE/

D1:NDIR N:
APPLE_II/                          76
APPLE_III/                        126
ATARI_8-BIT/                      604
COCO/                             268
COLECO_ADAM/                      164
commodore/                         46
Macintosh/                        198
tmp/                              106
boot.zx                           934
plato.tap                         26K
999+FREE SECTORS
```

Example XV.12. Adding to Network Prefix with NCD

```
D1:NCD N:ATARI_8-BIT

D1:NPWD
TNFS://APPS.IRATA.ONLINE/ATARI_8-BIT/

D1:NDIR N:
BBS/                               64
Comms/                            328
Databases/                        104
Demos/                             24
DOS/                              514
Educational/                       24
Games/                            764
Graphics/                          30
incoming/                          22
Internet/                         368
Languages/                        128
Mocks/                             80
Music/                            120
N-Demos/                           48
NOS/                               20
Novelties/                         92
PrintShop/                         82
Spreadsheets/                      58
Tests/                             18
TimeMgmt/                          48
Utils/                            178
WordProc/                         218
fnc-tools-doc.atr                 90K
fnc-tools.atr                     90K
fujinet-dosxl-tools.atr           90K
FujiNetToolsSpartaDOS.atr         90K
FujiNetToolsSpartaDOSSource.atr
90K
n-handler.atr                     90K
```

```
SDX449_ULTIMATE1MB_FUJINET.ATR 1000K
999+FREE SECTORS
```

Example XV.13. Using NCD with '..' for Parent Directory

```
D1:NPWD
TNFS://APPS.IRATA.ONLINE/ATARI_8-BIT/

D1:NCD N:..

D1:NPWD
TNFS://APPS.IRATA.ONLINE/
```

Example XV.14. Using NCD to Return to Root Directory

```
D1:NPWD
TNFS://APPS.IRATA.ONLINE/ATARI_8-BIT/Games/Homesoft/A/

D1:NCD N:/

D1:NPWD
TNFS://APPS.IRATA.ONLINE/
```

Example XV.15. Using NCD to reset path relative to Root Directory

```
D1:NPWD
TNFS://APPS.IRATA.ONLINE/ATARI_8-BIT/Games/Homesoft/A/

D1:NCD N:/ATARI_8-BIT/DOS/

D1:NPWD
TNFS://APPS.IRATA.ONLINE/ATARI_8-bit/DOS/
```

Returns

Error code of 0 on success, 1 on invalid parameters, otherwise the SIO error code is returned.

•••

…

```
...
```

Description

This command tells the FujiNet to reset itself. After waiting a few moments, the ATARI® is cold started.

Parameters

`...` …

`...` …

Examples

Example XV.16. ...

Returns

Error code of 0 on success, 1 on invalid parameters, otherwise the SIO error code is returned.

Colophon

This book was produced using the DocBook 5.1 XML schema, and XSL style sheets.

The source material for this book may be found on GitHub at the "fujinet-manuals" repository inside the FujiNet repository group:

https://github.com/FujiNetWIFI/fujinet-manuals/tree/main/fujinet_for_atari_users

For the printed format, the xsltproc tool was used to generate the intermediate FO format, and Apache FOP was used to process the intermediate format into PDF.

The HTML format is processed using OpenJade, and styled with CSS style sheets.

Index